

BIRKBECK, UNIVERSITY OF LONDON

Department of Computer Science and Information Systems

MSc Computer Science

Project Report

Timeline Visualization of Cultural Heritage Linked Data

Author: James Hill

Supervisor: Dr. Nigel Martin

2016

This report is substantially the result of my own work, expressed in my own words, except where explicitly indicated in the text. I give my permission for it to be submitted to the JISC Plagiarism Detection Service. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

This report documents the design and implementation of a desktop application that queries cultural heritage Linked Data and displays the results on dynamic timelines. The purpose is to create an application that cultural heritage researchers could use to query, analyse and store data about objects from multiple collections. The design of the application and choice of technologies used is explained with reference to how they satisfy the original specification. Challenges encountered and overcome during the development process are discussed in detail. The testing of the application is described. The report ends with an evaluation of the features that were implemented with recommendations for further development.

Supervisor: Dr. Nigel Martin

Table of Contents

1	Introduction	12
2	Background	13
2.1	Museum Collections Online	13
2.2	The Semantic Web	13
2.3	The CIDOC Conceptual Reference Model	14
2.4	Information Visualisation.....	14
2.5	Timelines	15
3	Specifications.....	17
3.1	Source Management, Querying and Timeline Tools.....	17
3.1.1	<i>Source Management Tool</i>	<i>17</i>
3.1.2	<i>Querying Tool</i>	<i>17</i>
3.1.3	<i>Timeline Tool</i>	<i>17</i>
3.2	Themed Division of Results.....	18
3.3	Administrative Tools	18
4	Trailer.....	19
4.1	Collection Interface.....	19
4.1.1	<i>Information Tab.....</i>	<i>19</i>
4.1.2	<i>Queries Tab.....</i>	<i>20</i>
4.1.3	<i>Entities Tab</i>	<i>20</i>
4.1.4	<i>Visualizer Tab.....</i>	<i>21</i>
4.2	Viewer Pane	22
4.3	Properties Pane.....	23
4.4	Source Management Tool.....	24
4.5	Output Window	24
4.6	Branding.....	25
4.6.1	<i>Splash Screen</i>	<i>25</i>
4.6.2	<i>Application Icons in Various Sizes</i>	<i>25</i>
5	Platform and Architecture.....	26
5.1	Java and Rich Client Platforms	26

5.2	Netbeans Platform Modules	28
5.2.1	Core Module	29
5.2.2	Source Module	31
5.2.3	Collections Module	32
5.2.4	Queries Module	34
5.2.5	Entities Module	35
5.2.6	Viewer Module	36
5.2.7	Dispatcher Module	36
5.2.8	Properties Module	36
6	Implementation	38
6.1	Technologies	38
6.1.1	Netbeans Platform	38
6.1.2	Netbeans IDE	38
6.1.3	Java	38
6.1.4	Apache Jena	39
6.1.5	Swing GUI Toolkit	39
6.2	Challenges	40
6.2.1	Unreliable Service from SPARQL Endpoints	40
6.2.2	Inconsistent Implementations of the CIDCO CRM	41
6.2.3	Writing the SPARQL Queries	42
6.2.4	Querying Years and Dimensions	43
6.2.5	Displaying the Best Division of Results	43
6.2.6	Implementing the TimeLines	45
7	Testing	46
7.1	Pre-Alpha Testing	46
7.1.1	Unit Testing	46
7.1.2	Debugging	47
7.2	Alpha Testing	47
7.3	Profiling	51

7.4	Beta Testing	52
7.4.1	Recruiting Testers	52
7.4.2	The Testing Process	53
8	Summary	54
8.1	Conclusions	54
8.2	Evaluation	55
8.3	Further Development.....	56
9	References.....	58
10	Appendix: User Manual.....	59
1	Introduction	61
2	Concepts.....	62
2.1	Sources.....	62
2.2	Queries.....	62
2.3	Entities	62
2.4	Collections.....	62
3	Installation	63
4	Getting Started.....	64
5	Managing Sources	65
5.1	New.....	66
5.2	Edit.....	67
5.3	Delete.....	67
6	Creating Collections	68
6.1	Information.....	68
6.2	Queries.....	69
6.3	Entities	70
6.4	Visualization	70
7	Properties.....	72
8	Output.....	73
9	Uninstallation.....	74
10	Tips.....	75
11	Appendix: Recruiting Testers.....	76

11.1	Email Description of the Application	76
12	Appendix: Code	78
12.1	Collections Module	78
12.1.1	<i>CollectionImpl.java</i>	78
12.1.2	<i>CollectionContainer.java</i>	81
12.1.3	<i>CollectionDisplayPanel.java</i>	83
12.1.4	<i>EntityDisplay.java</i>	87
12.1.5	<i>ScaleBuilder.java</i>	90
12.1.6	<i>TimeLine.java</i>	94
12.1.7	<i>TimeLineCollection.java</i>	100
12.1.8	<i>CollectionTopComponent.java</i>	107
12.1.9	<i>CollectionXMLParserImpl.java</i>	138
12.1.10	<i>CollectionXMLWriterImpl.java</i>	141
12.2	Dispatcher Module.....	145
12.2.1	<i>Dispatcher.java</i>	145
12.3	Entities Module	147
12.3.1	<i>Entities.java</i>	147
12.3.2	<i>ManMadeObject.java</i>	149
12.3.3	<i>PhysicalThing.java</i>	151
12.3.4	<i>EntitiesCollection.java</i>	157
12.4	Properties Module	159
12.4.1	<i>PropertiesTopComponent.java</i>	159
12.5	Queries Module	172
12.5.1	<i>QueryBuilder.java</i>	172
12.5.2	<i>QuerySettings.java</i>	173
12.5.3	<i>SPARQLTranslator.java</i>	175
12.5.4	<i>SPARQLQueryShell.java</i>	182
12.5.5	<i>QueriesCollection.java</i>	187
12.6	Sources Module	189

12.6.1	<i>SPARQLEndpoint.java</i>	189
12.6.2	<i>SourceCollection.java</i>	191
12.6.3	<i>SourceManagementTool.java</i>	194
12.6.4	<i>SourceManagerXMLParserImpl.java</i>	198
12.6.5	<i>SourceManagerXMLWriterImpl.java</i>	200
12.7	Viewer Module.....	202
12.7.1	<i>ViewerTopComponent.java</i>	202

Acknowledgments

I'd like to express gratitude to Dominic Oldman for his updates on the British Museum SPARQL endpoint availability and advice on writing SPARQL queries to extract CIDOC-CRM data.

Figures

FIGURE 1: 'A NEW CHART OF HISTORY' BY JOSEPH PRIESTLEY.	15
FIGURE 2: THE INFORMATION TAB.	19
FIGURE 3: THE QUERIES TAB.....	20
FIGURE 4: THE ENTITIES TAB.....	21
FIGURE 5: THE VISUALIZER TAB.	22
FIGURE 6: THE VIEWER PANE.....	22
FIGURE 7: THE PROPERTIES PANE.	23
FIGURE 8: THE SOURCE MANAGEMENT TOOL.	24
FIGURE 9: THE OUTPUT WINDOW.	24
FIGURE 10: A CLASS-STYLE DIAGRAM OF THE MODULE RELATIONS.	29
FIGURE 11: THE RELATIVE POSITION OF MODES WITHIN THE APPLICATION.	30
FIGURE 12: HEAP MEMORY CYCLING.....	51

Tables

TABLE 1: NETBEANS PLATFORM MODULES WITHIN THIS APPLICATION.	28
TABLE 2: THE .XML FILES WITHIN THE CORE MODULE.	31
TABLE 3: CLASSES WITHIN THE SOURCE MODULE.....	32
TABLE 4: CLASSES WITHIN THE COLLECTIONS MODULE.....	34
TABLE 5: CLASSES WITHIN THE QUERIES MODULE.	35
TABLE 6: CLASSES WITHIN THE ENTITIES MODULE.	36
TABLE 7: THE VIEWER MODULE.	36
TABLE 8: THE DISPATCHER MODULE.	36
TABLE 9: THE PROPERTIES MODULE.....	37
TABLE 10: SWING EXTENSIONS USED IN THIS PROJECT.	40
TABLE 11: DIMENSIONS USED TO FILTER THE RESULTS.	45
TABLE 12: RESULTS OF THE ALPHA TESTING, SECTION 1.....	49
TABLE 13: RESULTS OF THE ALPHA-TESTING, SECTION 2.....	50

Abbreviations

API	-	Application Programming Interface
AWT	-	Abstract Window Toolkit
CIDOC	-	ICOM's International Committee for Documentation
CLAROS	-	Classical Art Research Online Services
CPU	-	Central Processing Unit
CRM	-	Conceptual Reference Model
CRM-EH	-	Conceptual Reference Model – English Heritage
GUI	-	Graphical User Interface
ICOM	-	International Council of Museums
IDE	-	Integrated Development Environment
InfoVis	-	Information Visualization
JAR	-	Java Archive File
JVM	-	Java Virtual Machine
OWL	-	Web Ontology Language
RCP	-	Rich Client Platform
RDF	-	Resource Description Format
SPARQL	-	SPARQL Protocol and RDF Query Language
SWT	-	Standard Widget Toolkit
URI	-	Uniform Resource Identifier
W3C	-	World Wide Web Consortium

1 Introduction

Within the last ten years, museums, libraries and archives have started to publish their collection databases online. These databases contain information about the objects and documents held by the institutions: for example, the names, types, dimensions, current locations, date of creation, creators, and images of the objects.

Cultural heritage institutions have developed a unified methodology for storing this information in a queryable form; the CIDOC CRM, a semantic web ontology that models how cultural heritage researchers characterise objects. There are few applications that take advantage of this data, query, collect and analyse it to potentially produce novel insights for archivists, archaeologists and art collectors.

The goal of this project is to create a desktop application that allows the querying of multiple CIDOC CRM databases and displays the results on a graphical timeline. Objects returned from queries will be placed in relationships to one other; primarily chronologically but also through the filtering of objects into categories.

The following report outlines the history of the CIDOC CRM; describes the specifications for the application; describes user experience of the application; explains architectural and technical decisions taken during the design process; describe the testing of the application; and finally reflects upon the project and possibilities for further development of the application.

2 Background

Searching for objects held in museum collections is currently similar to using search engines. The CIDOC CRM has been developed to enable advanced querying of these databases; information visualization techniques may be used to present the data in novel and enlightening ways.

2.1 Museum Collections Online

Many museums, libraries and archives provide access to their collection data through search tools on their websites. For example, The British Museum¹ displays search results to the user as lists of photographs of the objects in the collection with text providing further details.²

Advanced options are often provided for sophisticated querying of multiple dimensions of the data; these features are familiar to users of standard search engines. Results can usually be sorted on a limited number of dimensions but relationships between objects can be interpreted only through textual analysis.

Larger sets of results may be displayed over multiple web pages with navigation buttons required to switch between them. The relationships between objects on different pages are obscured by this artificial division. The user experience of searching is also interrupted by delays in the loading time of pages.

Some museums, such as the Rijksmuseum, allow users to register and create personal galleries of objects from their collections. Such features are not common and the personal gallery a user creates is limited to the website and collection of single institutions.

2.2 The Semantic Web

The Semantic Web is a vision of the internet as a “web of data” that can be searched and analysed by machines as well as humans. [1] The set of interrelated online databases using semantic web technology is referred to as Linked Data. [2]

The World Wide Web Consortium (W3C³) has created three key technologies to enable the semantic web: the Resource Description Format (RDF) for defining a common data format;

¹ See https://www.britishmuseum.org/research/collection_online/search.aspx

² For example, see this search for a ‘jade pendant’:

http://www.britishmuseum.org/research/collection_online/search.aspx?searchText=jade+pendant

³ See <https://www.w3.org/Consortium/>

the Web Ontology Language (OWL) for organizing data around ‘ontologies’; and the SPARQL Protocol and RDF Query Language for querying Linked Data. [1]

2.3 The CIDOC Conceptual Reference Model

Working through CIDOC, a committee of the International Council of Museums (ICOM), cultural heritage institutions have designed a semantic web ontology that describes the structure of data they hold about collections. [3]

The CIDOC CRM ontology provides a framework for describing the relationships between events, objects and people.⁴ It defines relations between classes, properties and inheritance rules that can be encoded into programming languages. [3]

This CIDOC CRM enables integration of cultural heritage databases between institutions and across disciplines. The value of this data is enhanced when placed in relation with other data: researchers may create detailed and accurate global information and perform statistical analysis; developers can present the data in novel forms. [4]

A small number of institutions have published their collections online as CIDOC CRM Linked Data. These include the British Museum⁵, The Yale Centre for British Art⁶ and The Smithsonian American Art Museum⁷. Sophisticated querying of these databases is limited to the technologically proficient due to a dearth of accessible user interfaces and visualizations. [5]

2.4 Information Visualisation

Information Visualisation (InfoVis) uses computers to visually represent data in an interactive format that helps boost human cognition. [6] InfoVis tools can assist users make sense of large amounts of data:

“Visuals help understanding by acting as a frame of reference or as a temporary storage area for human cognitive processes. By providing a larger working set for thinking and analysis they become external cognition aids.” [5]

InfoVis tools were identified at an early stage of the CIDOC CRM project as a potential benefit of publishing cultural heritage collections in a semantic form.⁸ [4] Despite this, there

⁴ Appendix 1 includes examples of two subsets of the CIDOC CRM relations; spatial and temporal.

⁵ See <http://collection.britishmuseum.org/>

⁶ See <http://britishart.yale.edu/collections/using-collections/technology/linked-open-data>

⁷ See <http://americanart.si.edu/collections/search/lod/about/>

⁸ See M. Doerr and N. Crofts’ description of a simple website that presented a timeline of images of the Tower of Babel by the Renaissance artist Bruegel and his sons. [3]

chronological relationships between artefacts within collections should enhance understanding of those collections.

3 Specifications

The requirement of this project was the design of a desktop application that humanities researchers could use to remotely query CIDOC CRM databases and interact with a visual representation of the results to uncover relationships between objects. Three sets of objectives outlined during the proposal stage are summarised below.

3.1 Source Management, Querying and Timeline Tools

Three tools were considered essential for a minimum viable application; a source management tool, a tool for creating queries, and a timeline tool for displaying the returned results.

3.1.1 Source Management Tool

This tool would allow the user to control their personal list of SPARQL endpoints. The user would be able to create, edit and delete sources.

3.1.2 Querying Tool

This tool would allow the user to create SPARQL queries for CIDOC CRM data within a simplified graphical interface. Search parameters for interesting dimensions of the CIDOC CRM would be manipulated through text fields and other components. The tool would be capable of querying multiple databases and returning the results as a single collection.

3.1.3 Timeline Tool

The timeline tool would present results returned from queries as images on an interactive visual timeline so that chronological relationships between objects would be immediately apparent to the user. Images would be enlarged by pointing at them. Clicking on images would display further information about the objects in a second window. The user would be able to zoom into particular time periods.

3.2 Themed Division of Results

A collection of objects would be filterable on multiple dimensions so that the user would be able to divide the results into themed timelines. These filtered timelines would represent interesting dimensions in the data such as the type of object, the materials of construction, or the artist who produced the object.

The filters would also include functions to assist the user identify the most interesting sub-collections of objects, both through sorting of the TimeLines and through an algorithm that attempts to choose the most interesting dimensions to display to the user. The user would be able to override these functions and choose their own dimensions of interest, and would be able to drill into the data and filter subsets by further criteria.

3.3 Administrative Tools

Further tools were suggested as extras that could be implemented if the timeline tool was completed early. Saving and loading of queries and results were proposed to allow work to continue over multiple sessions. The export of timelines and object information would allow users to present their results in other contexts such as presentations or reports.

4 Trailer

The application has been designed as a Windows-type program with a menu bar, sub-windows and tabbed panes that would be familiar to anyone who has worked with common office applications such as those within the Microsoft Office suite.

4.1 Collection Interface

Queries and the objects they return are held in projects (named 'Collections') that user can open, close, save and delete. The Collection Interface is a window that can be instantiated at the centre of the application workspace to allow users to interact with the Collections. Multiple Collection Interface windows can be open concurrently.

The Collection Interface contains four tabs, each containing tools enabling users to annotate, manipulate and view Collections.

4.1.1 Information Tab

A requirement of the application was that work could continue over user sessions. The Information Tab contains two elements: a text field to record a name that enables the Collection to be identified throughout the application; and a text area for storing notes about the Collection. These allow the user to identify and recall Collections produced in previous sessions.

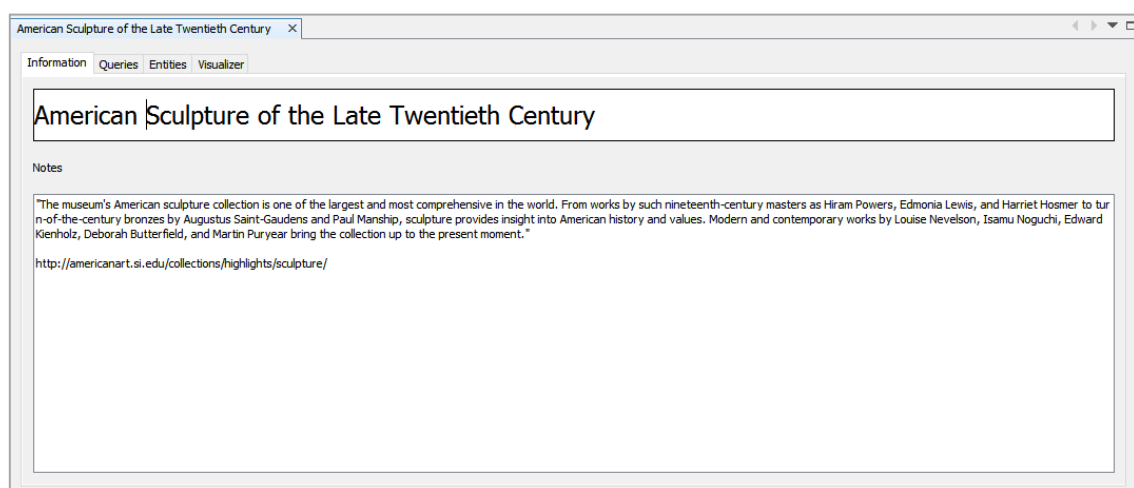


Figure 2: the Information Tab.

4.1.2 Queries Tab

Two requirements of the application impact the design of the Queries Tab: firstly, the user may query multiple sources concurrently; and secondly, the creation of queries must be simplified so that users do not need technical knowledge to program their own SPARQL queries.

To meet the requirement that users may query multiple sources concurrently each Collection must be capable of holding multiple queries. The user can add and delete queries to a collection and they are represented by a table on the left side of the Queries Tab.

To meet the requirement that the creation of a query is simplified, a Query Builder tool sits at the right side of the Queries Tab. Queries are built by the user interacting with text fields, combo boxes and check boxes to choose options.

Name	Last Run	Query Type
1950s		SPARQL Endpoint
1960s		SPARQL Endpoint
1970s		SPARQL Endpoint
1980s		SPARQL Endpoint
1990s		SPARQL Endpoint

Figure 3: the Queries Tab.

4.1.3 Entities Tab

Within the application, 'Entities' are objects held within cultural heritage institutions and returned as results of queries. An example Entity is the Rosetta Stone held at the British Museum.¹² The application requires that Entities may be ordered to facilitate their understanding in context and that users may view collections of objects without the interruption of page dividers.

¹² The Rosetta Stone is an ancient Egyptian stela with a decree carved in three languages whose discovery during Napoleon's invasion of Egypt led to the decipherment of hieroglyphics.

The Entities Tab fulfils these requirements by providing a table where each row represents an object. Columns hold details about the objects and can be sorted so that the user can view them in different relations to each other. This tab provides a supplementary and alternative view to the main visualization of objects, the Visualizer Tab.

Identifier	Name	Query	Source	Image	Year	Material	Type	Technique	Creator
1995.36.2	#8 Study in Stone	1980s	http://edan.si.ed...	http://americanar...	1989	glass, granite, an...	Sculpture		Jack A. Schmidt
2010.31	...my mom's dres...	1990s	http://edan.si.ed...	http://americanar...	1999	cardboard, pins, ...	Sculpture		Gale Jamieson
1991.176	17th Summer	1970s	http://edan.si.ed...	http://americanar...	1974	acrylic on wood	Sculpture		Anne Dean Truitt
2005.5.2	35 Year Portrait	1980s	http://edan.si.ed...	http://americanar...	1986	glazed ceramic	Sculpture		Robert Carston A...
1980.137.87	A Very Thick Sum...	1960s	http://edan.si.ed...	http://americanar...	1969	mixed media: pain...	Sculpture		Sam Richardson
1985.30.37	Adrian	1950s	http://edan.si.ed...	http://americanar...	1957	bronze on wood b...	Sculpture		Oronzio Maldarelli
1995.5	Arabian Seasons	1990s	http://edan.si.ed...	http://americanar...	1994	glass, stone, woo...	Sculpture		Therman Statom
1988.93A-C	Baby Dragon	1980s	http://edan.si.ed...	http://americanar...	1985	mixed media (curr...	Sculpture		Barton Benes
1987.18.1	Beaver	1960s	http://edan.si.ed...	http://americanar...	1963	plaster	Sculpture		Zorach Samovich
1968.136	Black and White	1960s	http://edan.si.ed...	http://americanar...	1964	wood: birch	Sculpture		Louis Schanker
1995.11	Buffalo Dance	1980s	http://edan.si.ed...	http://americanar...	1983	Indiana limestone	Sculpture		Allan C. Houser
2014.63	Candlestick	1970s	http://edan.si.ed...	http://americanar...	1979	steel	Sculpture		Albert Raymond ...
1991.158.4	Cane with Alligato...	1990s	http://edan.si.ed...	http://americanar...	1991	carved and paint...	Sculpture		Tim Lewis
1991.158.5	Cane with Man an...	1990s	http://edan.si.ed...	http://americanar...	1991	carved and paint...	Sculpture		Denzil Goodpaster
1991.158.1	Cane with Rattles...	1980s	http://edan.si.ed...	http://americanar...	1982	wood; solvent ba...	Sculpture		Ben Miller
1995.85.2	Chang	1980s	http://edan.si.ed...	http://americanar...	1989	carved basswood...	Sculpture		Robert D. Brady
1986.6.38	Cob I	1980s	http://edan.si.ed...	http://americanar...	1980	carved wood, lea...	Sculpture		Nancy Grossman
1986.65.257	Crucifix	1970s	http://edan.si.ed...	http://americanar...	1971	carved wood with...	Sculpture		José Domingo Mo...
1966.110.57	D.W. McMillen	1950s	http://edan.si.ed...	http://americanar...	1954	bronze	Sculpture		Anthony DeFranc...
1998.94A-D	Devoción de Nue...	1990s	http://edan.si.ed...	http://americanar...	1998	gesso and natural...	Sculpture		Charlie Carrillo
1967.93.61	Drug Industry An...	1960s	http://edan.si.ed...	http://americanar...	1963	bronze	Sculpture		Joseph Renier

Figure 4: the Entities Tab.

4.1.4 Visualizer Tab

The Visualizer Tab fulfils a central requirement of the application; an interactive visualization of objects that can assist researchers analyse collections more insightfully than through a lists of results. Objects would be represented by image and could be ordered along various dimensions.

The Visualizer Tab displays objects using the concept of 'Timelines'. Each TimeLine contains a subset of objects and the number of TimeLines and the objects they contain is determined by dimensions chosen by the user from option boxes. This fulfils the requirement that users may dynamically reorder the results to make sense of the relationships between objects.

Objects are displayed within Timelines as representative images downloaded from the online collection databases at their chronological position along a scale representing the time period encompassing all displayed objects. Holding the mouse pointer over an image will display an enlarged copy. This fulfils the requirement that users can identify objects by image alone.

There is a 'Save' button which when pressed saves all details of the Collection to an .xml file that can be reloaded into the application on startup.

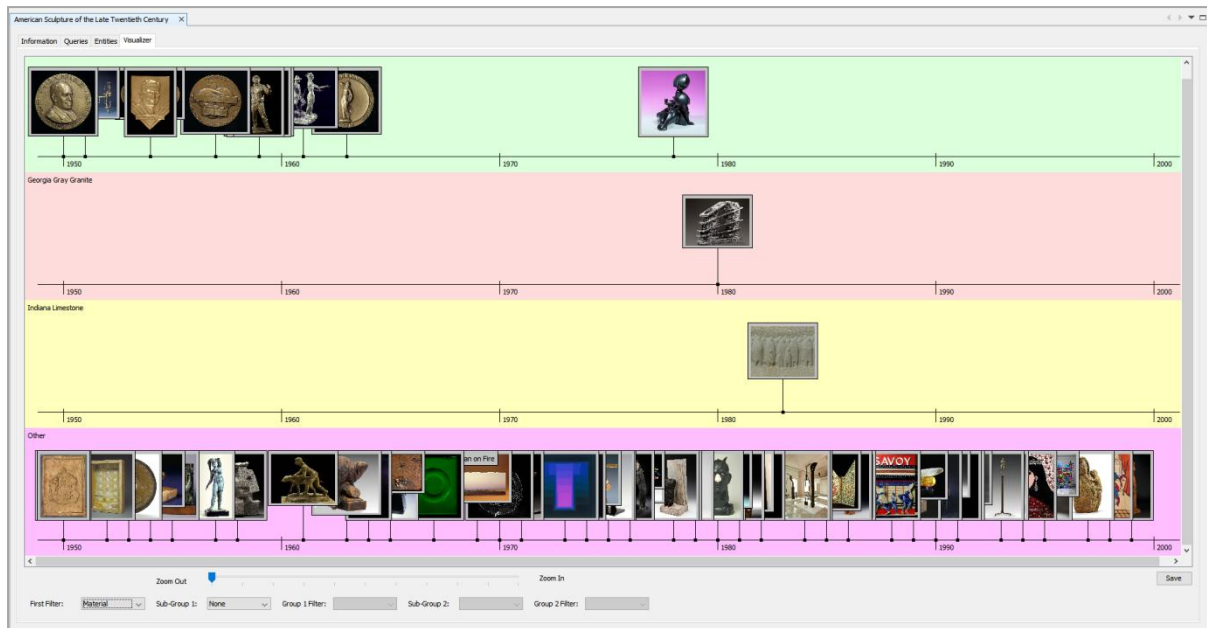


Figure 5: the Visualizer Tab.

4.2 Viewer Pane

The application required that ideally the user may save and load queries and results so that work on collections can continue between sessions. The Viewer Pane displays a list of Collections created by the user. These Collections can be opened or deleted by the user. If a Collection is saved it will be reloaded into the Viewer Pane when the application is next opened. This fulfils the requirement that users may continue work over sessions and also ensures that they may work on multiple projects simultaneously.

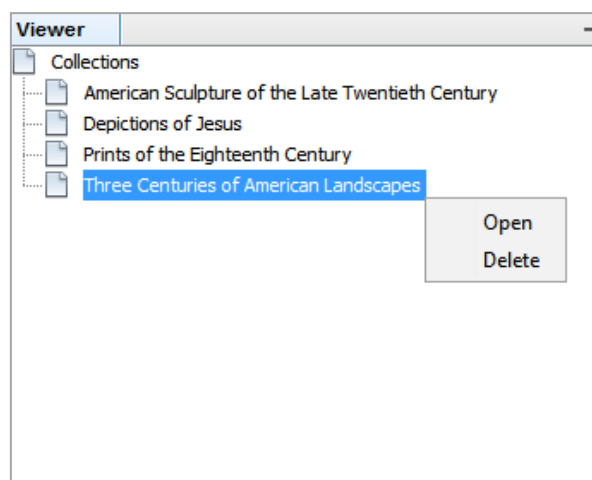



Figure 6: the Viewer Pane.

4.3 Properties Pane

A requirement of the project is that detailed information about objects must be available to the user once identified visually. This has been implemented through the addition of a Properties Pane that displays an enlarged image and textual information about any object that has been mouse-clicked in the Visualization Tab.

Properties



Creation Year:

1807

Identifier:

PPA81648

Name:

The head of the poll, or the Wimbledon

Source:

<http://collection.britishmuseum.org/spa/>

Query:

1800s

Depicts:

Jesus Christ

Consists of:

paper

Is of type:

print

Technique:

Creator:

Charles Williams

Further details:

<http://collection.britishmuseum.org/id/o>

Description:

(For description see other impression).
May 1807

Hand-coloured etching

Commentary:

NB The register number stamped on the print is a duplicate (the number also belongs to a satirical print). I have therefore added an asterisk to the number.

Figure 7: the Properties Pane.

4.4 Source Management Tool

The application requires that users may create, edit and delete SPARQL endpoints. This has been implemented through a Source Management Tool; a window with a table displaying available endpoints and buttons to fulfil commands.

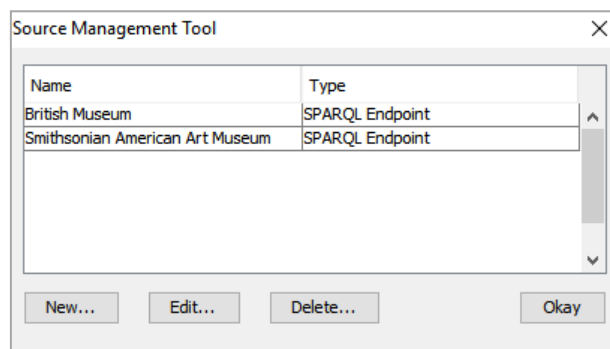


Figure 8: the Source Management Tool.

4.5 Output Window

The Output window is an optionally displayed window that displays text recording actions taken by the user or data being downloaded.

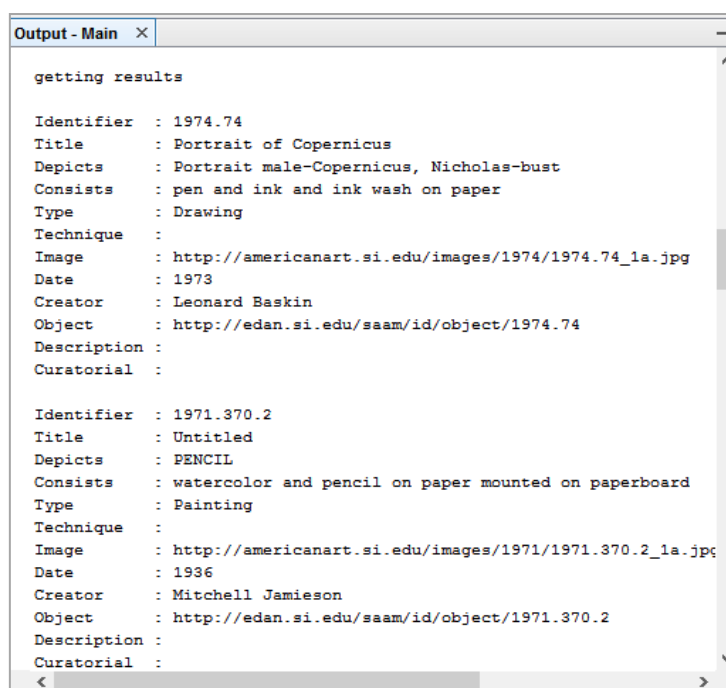


Figure 9: the Output Window.

4.6 Branding

Graphics were created and added to replace the default Netbeans Platform 'splash screen' and application icons.

4.6.1 Splash Screen



4.6.2 Application Icons in Various Sizes

A symbolic representation of a museum was chosen as an icon.

16 x 16



32 x 32



48 x 48



5 Platform and Architecture

The specification requires a desktop application that users with limited computer literacy should be capable of learning to use quickly based on their knowledge of common workplace software. The application is therefore built around a main window with a menu bar and numerous internal windows requiring communication between the components.

Building an entire application with all of these elements working together from scratch would be a time-consuming task likely to be unachievable within the timeframe permitted for this project. Research in the project planning stage however revealed that there are tools that provide a base for desktop applications called Rich Client Platforms (RCPs).

RCPs are collections of pre-written and tested components that developers can harness for their own applications. A typical RCP has pre-existing features such as a modular structure, lifecycle management, a file system, a windows system, and a user-interface toolkit, among others, that developers can hook their applications into. The developer is then free to concentrate on the essentials of their application without needing to program basic functions common to all applications.

RCPs are tied to specific languages, provide internal structures that determine the high-level organization of the code, and are sometimes linked to other tools that simplify the development process. The choice of RCP is closely linked to other important choices that a developer must make during the design and implementation process. This application has been built upon the foundation of an RCP and the choice that was made is discussed before further details of the architecture and implementation are explained.

5.1 Java and Rich Client Platforms

Two prominent RCPs currently available for desktop application programming with ongoing community development and support are Java language RCPs: the Eclipse RCP and the Netbeans Platform. Both are popular and have been used for a large number of desktop applications.¹³ There were no other suitable choices for RCPs in other programming languages so the use of Java was determined by the initial requirement that a desktop application be produced in a short space of time.

The RCPs and Java are well suited to fulfil the requirements of the application due to their support for Graphical User Interface (GUI) code. Java has a number of APIs available for programming rich GUIs: AWT, Swing, SWT and JavaFX are well known examples. Each of the Java RCPs is built around a different Java GUI API: the Eclipse RCP is built around the IBM

¹³ A sample of Netbeans Platform applications can be found here:
<https://platform.netbeans.org/screenshots.html>

designed SWT API and the Netbeans Platform is built around the Swing API which is a native Java API designed by Oracle.

There are advantages to using the GUI API that each of the RCPs is built around. In the case of the Netbeans Platform, if the project is designed through the Netbeans IDE the developer may use the inbuilt Swing GUI Builder which allows for drag and drop GUI design.

Given the close integration between the RCPs and the GUIs they support, the choice of RCP is also heavily influenced by the choice of GUI. The following passage from the Eclipse wiki quite succinctly describes the differences between the choices:

“SWT and Swing are different tools that were built with different goals in mind. The purpose of SWT is to provide a common API for accessing native widgets across a spectrum of platforms. The primary design goals are high performance, native look and feel, and deep platform integration. Swing, on the other hand, is designed to allow for a highly customizable look and feel that is common across all platforms.”¹⁴

For the purpose of the current application, the aesthetic choice outlined by the above passage is irrelevant; the application is not being built initially for development as a cross-platform product with a strong brand image as a commercial product would be. However, the difference between the design goals of the two APIs does have a practical impact on the choice of RCP.

Swing components are sometimes called ‘lightweight’ components. They are written in pure Java code and run entirely through the Java Virtual Machine (JVM). This ensures that the look and feel of the components is common across all platforms. This provides simplicity for the developer as they only need concern themselves about the initial writing of the components; they can be assured that all platforms will support them.

The SWT components are on the contrary referred to as ‘heavyweight’ components because they wrap around platform-native GUI components. This ensures that the look and feel of components matches the platform being used but has some negative aspects: firstly, the GUI is exposed to platform specific bugs; secondly, a platform specific SWT library must be distributed with the application for each platform the developer wishes to support. SWT components are also limited by the features of the platform specific components that they wrap around and are therefore less adaptable than the Swing components.

For the purpose of this application, the simplicity of the Swing components, particularly the requirement that no understanding of platform specific details would be necessary to produce the GUI, and the relative extensibility of the components, allowing them to be easily adapted for the visualization component of the application, favoured Swing as the choice of GUI.

¹⁴ http://wiki.eclipse.org/FAQ_Is_SWT_better_than_Swing%3F

Swing as the choice of GUI therefore gave some weight to Netbeans Platform as the choice of RCP given the close integration between Netbeans Platform, the Netbeans IDE and the Swing GUI Builder. It is possible to build Swing applications through Eclipse but the integration between the IDE and GUI API is not ‘out of the box’ and requires initial setup that is unnecessary with the Netbeans Platform and IDE.

It is possible to build a Netbeans Platform application without the Netbeans IDE but this may confront the developer with compatibility difficulties they would otherwise never encounter. Fortunately, the Netbeans IDE has other features that commend it; most importantly for this project, Java JUnit testing, Git integration, and integrated debugging and profiling tools. The choice of technologies for the development of the application was therefore the combination of Java, Netbeans IDE, Netbeans Platform and Swing API.

5.2 Netbeans Platform Modules

The Netbeans Platform is constructed around Modules; these are Java Archive (JAR) files written to interact with the Netbeans APIs. They can be installed and removed independently within an application and should be designed to be as independent from one another as possible. This application is constructed around the modules in the following table:

Module Name	Description
TimelineVisualizerCore	Code affecting the application setup and structure.
TimelineVisualizerCollections	‘Collection’ object classes.
TimelineVisualizerDispatcher	Code controlling communication with SPARQL endpoints.
TimelineVisualizerEntities	‘Entities’ object classes.
TimelineVisualizerProperties	Code for the Properties window.
TimelineVisualizerQueries	‘Query’ object classes.
TimelineVisualizerSources	‘Source’ object classes.

Table 1: Netbeans Platform modules within this application.

Netbeans Platform Modules communicate with each other in two ways: either through directly exposing internal packages as an API; or through a Lookup class, a general registry that allows objects to be placed into and collected from a map-like structure accessible to all modules. These two methods were both used in this application as appropriate.

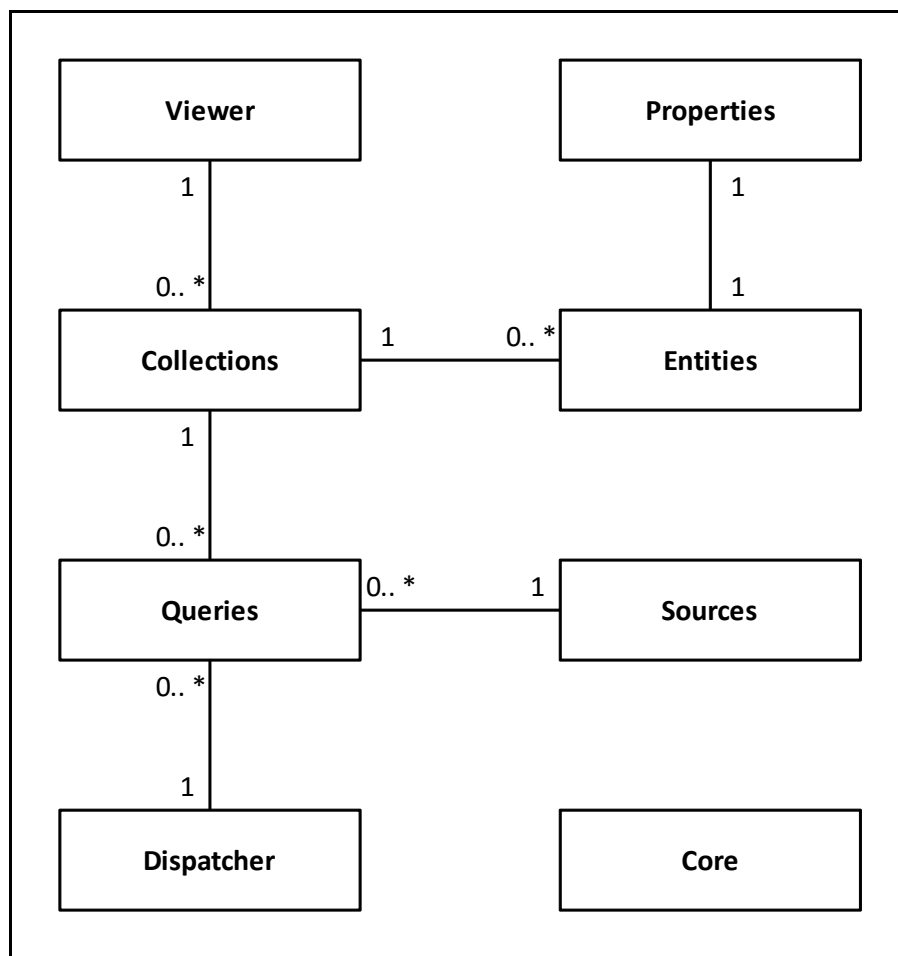


Figure 10: a class-style diagram of the module relations.

5.2.1 Core Module

The core module stores settings that apply to the entire application. These settings cover two elements of the application GUI: menu and control bars; and the positioning of internal windows.

The Netbeans Platform is built upon pre-existing modules that form the basis of a new application. Modules can be removed if an application does not require them. Along with all of the hidden code that allows modules to interact with each other, these modules also provide many menus and control bars. Few of these menus may be desirable in an application although functions and classes in the modules they derive from may be required.

The appearance of menus, control bars and windows within the Netbeans Platform are modified through 'layers'. A layer is an .xml file describing the state and properties of these GUI components. This application has an .xml file modified so that only the absolute

minimum of menu items from the Netbeans Platform default appear within the menu bar. All control bars were removed as they extraneous. This simplifies the interface the user interacts with.

Windows within a Netbeans Platform application appear within tabbed containers called 'modes' which are divided by 'splitters'. Modes are linked to positions within the application such as 'top side' or 'right side'; they can also be assigned to sliding positions so that they can be minimized to a bar along one side of the main application window.

Figure 11 shows the location of modes and their components within this application. The Viewer Pane appears on the left side with the Output Window below it. Multiple Collection Interface windows can appear as tabs within the 'Collections' mode. The Properties pane appears in the 'Properties' mode at the right side.

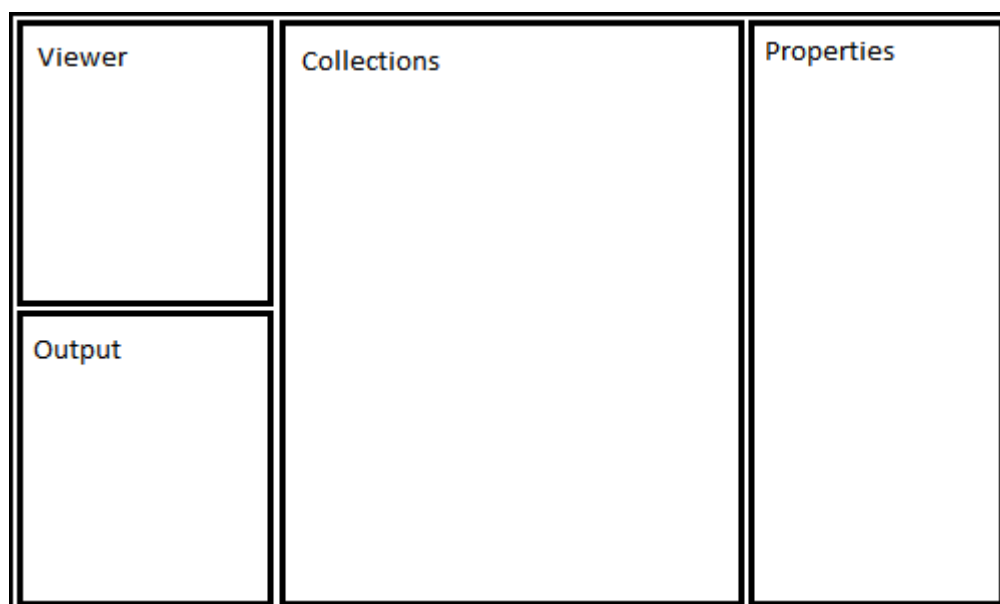


Figure 11: the relative position of modes within the application.

The Netbeans Platform has many default modes; these custom modes were programmed and the default mode removed so that the various windows within the application are positioned and sized correctly. This removes the risk of users accidentally dragging and dropping components into odd locations within the main window and provides a visually consistent workspace.

Component	Name	Notes
Package	org.jghill.timelinevisualizercore	
Netbeans.xml	layer.xml	Defines the menu and control bar.
Netbeans.xml	collectionsWsmode.xml	Defines 'Collections' window mode.
Netbeans.xml	outputWsmode.xml	Defines 'Output' window mode.
Netbeans.xml	propertiesWsmode.xml	Defines 'Properties' window mode.
Netbeans.xml	viewerWsmode.xml	Defines 'Viewer' window mode.

Table 2: the .xml files within the Core module.

5.2.2 Source Module

A requirement of the project is that the user may query multiple CIDOC CRM databases and display the combined results in order to enable the emergence of new connections between objects that would not be obvious through separate searches. As the number of CIDOC CRM implemented databases is likely to expand it was an essential requirement that the user can manage their own list of SPARQL endpoints.

This application uses the concept of 'Sources' to represent the databases that can be queried by the user. SPARQL endpoints require one Uniform Resource Identifier (URI) to identify their location on the internet and another to identify the version of the CIDOC CRM ontology the data has been encoded in.

CIDOC CRM databases are unlikely to change their location or encoding often and new databases should only occasionally become available so the user should not regularly need to interact with Sources. The Source Management Tool used to manipulate Sources within the application is therefore accessed through the menu bar.

The Source module contains the classes implementing the Source Management Tool and is divided into three packages: one defines the Source classes and organizational structures to contain them; another defines the GUI through which the user interacts with Sources; and the third defines classes for the saving and loading of Sources.

The Source Collection class is a singleton holding a single container of Sources for each instance of the application. This ensures there are never two collections of Sources existing simultaneously and that all Sources are placed within the same collection. This singleton is exposed through the API to other modules so that Sources can be used when the user defines queries.

The Sources are represented by an interface which can be implemented by classes representing different source types. The application currently only implements a class for SPARQL endpoints but may be extended to include other databases such as files held on a

local disk drive or network connection. This would enable the user to work from a backup file if their connection to a SPARQL endpoint failed.

The GUI package defines a set of Swing windows and panes through which the user interacts with the Source Collection to manipulate Sources. The windows can be extended with interchangeable panes that could be customized for different types of Source.

The final package contains interfaces and implementations for the writing and parsing of xml documents recording the Sources entered by a user. This allows Sources to persist over user sessions.

Component	Name	Notes
Package	org.jghill.timelinevisualizersources	
Class	SPARQLEndpoint.java	An extension of Source.java.
Abstract Class	Source.java	Represents any source of data.
Class	SourceCollection.java	Holds all Sources available.
Class	SourceTableModel.java	Defines a table structure for Sources
Package	org.jghill.timelinevisualizersourcesgui	
Swing GUI	SPARQLEndpointPanel.java	
Interface	SourceDetailsPanel.java	
Swing GUI	SourceManagementEdit.java	
Swing GUI	SourceManagementNew.java	
Class	SourceManagementOpen.java	
Swing GUI	SourceManagementTool.java	The main window.
Package	org.jghill.timelinevisualizersourcesxml	
Interface	SourceManagerXMLParser.java	Defines methods for parsing XML
Class	SourceManagerXMLParserImpl.java	Implements the XML parser interface
Interface	SourceManagerXMLWriter.java	Defines methods for writing XML
Class	SourceManagerXMLWriterImpl.java	Implements the XML writer interface

Table 3: classes within the Source module.

5.2.3 Collections Module

The Collections Module holds all classes relevant to the Collections and Collections Interface. Both the Querying Tool and the Timeline Tool identified in the specifications are instantiated within the Collections Interface. All Collections within the application are held in the CollectionsContainer class which is a singleton object of which there is only one instance.

The Collections Interface is a complex GUI tool with multiple tabs, text fields, panes, buttons, checkboxes and combo boxes. It is an extension of a class named 'TopComponent' that represents windows within the Netbeans Platform that can be assigned to modes. The Collections Interface is assigned to the Collections mode and multiple instances can be instantiated simultaneously.

Each Collection object contains a Java collection for holding queries. Queries are added to and deleted from these collections through an interface within the Collections Interface object. GUI components within the Collections Interface allow the user to define their choices for a query and these are passed to a new Query object, fulfilling the requirement that the construction of queries is simplified and abstracted from raw SPARQL code. The collection of Queries is passed to a Dispatcher class where they are run in turn thus fulfilling the criteria that multiple sources of data may be queried simultaneously by the user.

The Visualization Tab consists of a Collection Display Panel class that can hold multiple instantiations of the TimeLine class. The objects returned from queries are filtered by variables chosen by the user through combo-boxes at the bottom of the tab. The filtered results are displayed chronologically along the TimeLines as instantiations of the EntityDisplay class. This fulfils the criteria that results from queries are displayed chronologically and may be filtered along multiple dimensions.

Objects are displayed on TimeLines through an EntityDisplay class that accepts a ManMadeObject class and represents it visually as a selectable panel containing either an image or the name of the object. When selected, details of an object are passed into a global Lookup where they can be identified by code within the Properties Pane.

Component	Name	Notes
Package	org.jghill.timelinevisualizercollections	
Interface	CanDelete.java	Represents an action
Interface	CanOpen.java	Represents an action
Interface	Collection.java	Represents an action
Class	CollectionImpl.java	
Package	org.jghill.timelinevisualizercollections.container	
Class	CollectionsContainer.java	Contains all Collections
Package	org.jghill.timelinevisualizercollections.display	
Class	CollectionDisplayPanel.java	
Class	CollectionDisplayPanelBeanInfo.java	
Class	Colours.java	Defines colours used in TimeLines
Class	EntityDisplay.java	Wraps around an Entity
Class	ScaleBuilder.java	
Class	TimeLine.java	
Class	TimeLineBeanInfo.java	
Class	TimeLineCollection.java	
Package	org.jghill.timelinevisualizercollections.gui	
Class	CollectionNew.java	An action
Swing GUI	CollectionTopComponent.java	The main window
Class	EntityTableModel.java	
Class	QueryTableModel.java	
Package	org.jghill.timelinevisualizercollections.node	
Class	CollectionChildren.java	
Class	CollectionNode.java	
Class	ViewerDeleteAction.java	
Class	ViewerOpenAction.java	
Package	org.jghill.timelinevisualizercollectionxml	
Interface	CollectionXMLParser.java	
Class	CollectionXMLParserImpl.java	
Interface	CollectionXMLWriter.java	
Class	CollectionXMLWriterImpl.java	

Table 4: classes within the Collections module.

5.2.4 Queries Module

The QueryShell abstract class exists which is extended by the SPARQLQueryShell class; this design allows for the possibility of different query types to be implemented if the application is extended to support other data sources in future.

Information representing the options a user chooses within the Query Builder Tool is stored in instantiations of the QuerySettings class and passed to a singleton QueryBuilder object which decides how to build a query. This is an example of the object-oriented Command pattern with the QuerySettings object as the command object.

Queries are constructed from the information contained within QuerySettings objects by translation classes invoked by the QueryBuilder. The QueryTranslator interface may be implemented by multiple classes although the current version of the application only has an implementation for SPARQL queries. The SPARQLTranslator class contains the complex code that transforms the user's query options into a SPARQL Query.

Component	Name	Notes
Package	org.jghill.timelinevisualizerqueriesbuilder	
Class	QueryBuilder.java	
Class	QuerySettings.java	Contains settings defined by the user
Interface	QueryTranslator.java	
Class	SPARQLTranslator.java	Converts QuerySettings to SPARQL
Package	org.jghill.timelinevisualizerqueries	
Abstract Class	QueryShell.java	
Class	SPARQLQueryShell.java	

Table 5: classes within the Queries module.

5.2.5 Entities Module

Entities within the application represent objects held in institutional collections that have been returned by a query, or collections of these objects. The Entities class is an abstract class that is extended by both the EntitiesCollection class that may contain further Entities and the PhysicalThing abstract class.

This is an implementation of the object-oriented Composite pattern with the intention of allowing trees of objects to be constructed from multiple queries. The application currently only allows simple structures of two layers but this pattern allows for future development to implement commands allowing users to combine multiple EntitiesCollections into new Collections.

The PhysicalThing abstract class has been designed to be extended by multiple classes. The current application only extends this with the ManMadeObject class but the CIDOC CRM also allows for biologically derived objects (such as fossils). The PhysicalThing abstract class was not extended to include biological objects in this version of the application because no natural history institutions currently publish their collections as Linked Data.

Component	Name	Notes
Package	org.jghill.timelinevisualizerentities	
Abstract Class	Entities.java	
Class	ManMadeObject.java	
Abstract Class	PhysicalThing.java	
Package	org.jghill.timelinevisualizerentitiescollection	
Abstract Class	EntitiesCollection.java	

Table 6: classes within the Entities module.

5.2.6 Viewer Module

The Viewer module contains a singleton ‘TopComponent’ implementing the Viewer Pane and displaying the Collections held within the CollectionsContainer singleton.

Component	Name	Notes
Package	org.jghill.timelinevisualizerviewergui	
Swing GUI	ViewerTopComponent.java	

Table 7: the Viewer module.

5.2.7 Dispatcher Module

This module contains a singleton Dispatcher class that receives Queries from a Collection, runs the Queries and returns the resulting Entities to the Collection.

Component	Name	Notes
Package	org.jghill.timelinevisualizerdispatcher	
Swing GUI	Dispatcher.java	

Table 8: the Dispatcher module.

5.2.8 Properties Module

The Properties module contains a singleton ‘TopComponent’ that implements the Properties Pane. It contains a listener object attached to the Netbeans Platform generic Lookup which reacts to an Entity object being placed into the Lookup. The Entity object is

extracted and data from the object is displayed within the image panel and text fields within the Properties Pane. This fulfils the criteria that an object being viewed by the user will display more detailed information when mouse-clicked.

Component	Name	Notes
Package	org.jghill.timelinevisualizer.properties	
Swing GUI	PropertiesTopComponent.java	

Table 9: the Properties module.

6 Implementation

The design of the application and choice of technologies went hand-in-hand for this application due to the structures of the Netbeans Platform that were exploited. The initial modules written were those necessary to fulfil the requirement of a minimal viable application: the Source, Queries, Entities, Collections and Dispatcher modules. These were not completed at once but were expanded as necessary. The Viewer and Properties modules were added after the concept was proven by a working version.

6.1 Technologies

The core technologies were decided along with consideration of the design and the choice of RCP. There was however a choice of packages for manipulating SPARQL queries and some consideration of different possible technologies for implementing the GUI and interactive TimeLines.

6.1.1 Netbeans Platform

The Netbeans Platform provided the foundation for the application; features such as a modular structure, lifecycle management, a file system, a windows system, and a user-interface toolkit simplified the process of building a complex application. This has been discussed in detail in section 5 so will not be repeated here.

6.1.2 Netbeans IDE

The Netbeans IDE is a tool for developing Java applications and is integrated tightly with the Netbeans Platform. It has a Swing graphical API drag-and-drop design tool allowing for rapid development of GUIs, inbuilt support for the JUnit testing framework and Git version control system, and debugging and profiling tools. The reasons for choosing this IDE were also discussed in detail in section 5 so will not be repeated.

6.1.3 Java

The Java programming language is an object-oriented language released in 1995 and is amongst the most popular programming languages in use.¹⁵ It is the language of the Netbeans Platform and the main language supported by Netbeans IDE. The existing and

¹⁵ More information about Java here: <https://www.java.com/en/>

supported RCPs for desktop applications are written for Java so the choice of language was largely determined by the decision to use a RCP.

6.1.4 Apache Jena

Apache Jena is a suite of Java packages that manage the creation, dispatch and reception of results from SPARQL queries.¹⁶ An alternative SPARQL query management suite named Sesame Query Builder is available but does not support the most recent version of SPARQL (1.1) and appears to be significantly less well documented. Therefore the Apache Jena package was chosen for the implementation of SPARQL querying within this application.

6.1.5 Swing GUI Toolkit

An investigation into existing graphical packages that might provide a data visualization fulfilling the requirements of the application was not successful. Many packages exist with the ability to create dozens of types of charts but none fulfilled all of the following criteria:

- Display objects as images.
- Enlarge images when pointed at.
- Supplement images with meta-text.
- Place images on timelines.
- Support multiple timelines simultaneously.
- Provide zoom and scroll functions for the timelines.
- Provide an ability to 'drill-down' into a timeline.

The most suitable tools were the Jaret Timebars Component¹⁷ and TimeFlow¹⁸ libraries but these were unsatisfactory because they did not support the display of images and focussed on smaller expanses of time than necessary for this application.

In the absence of a suitable existing tool the visualization was implemented through extensions of the Swing GUI to build a custom timeline tool. The relative simplicity of the criteria, the integration of Swing into Netbeans IDE and RCP, and the use of Swing components throughout the GUI were reasons not to introduce the complexity of importing a further Java graphics package such as JavaFX.

Table 10 lists Swing components used or extended to create custom components for the visualization.

¹⁶ More information here: <https://jena.apache.org/>

¹⁷ See here: <http://www.jaret.de/timebars/index.html>

¹⁸ See here: <https://github.com/FlowingMedia/TimeFlow/wiki>

Swing Component	Extensions or Uses in this Project
JPanel ¹⁹	Boxes representing Entities and holding images or descriptive text.
JLayeredPane ²⁰	The TimeLine bars on which Entities are displayed.
JComboBox ²¹	Option boxes for filtering results into different themes.
JLabel ²²	Year numbers on the TimeLine scale, titles of the TimeLines.

Table 10: Swing extensions used in this project.

6.2 Challenges

Difficulties in implementing the application clustered around three problem areas: consistently extracting data from CIDOC CRM databases; enabling drill-down exploration of collections of objects through filters; and displaying zoom-able and scrollable timelines.

6.2.1 Unreliable Service from SPARQL Endpoints

Only a small number of cultural heritage institutions currently host CIDOC CRM format databases with access via SPARQL endpoints. Institutions I discovered are:

- The British Museum²³
- The Smithsonian American Art Museum²⁴
- The Yale Center for British Art²⁵
- The Archaeology Data Service (UK)²⁶
- CLAROS²⁷

The CLAROS website has a broken link on the homepage to their ‘Open Data’. The host of CLAROS, the Oxford e-Research Centre, was notified and a technical problem was discovered with the Linked Data server. This problem was not resolved at the time of writing this report so CLAROS data was not tested with this application.

The British Museum SPARQL endpoint was only intermittently available for many weeks and usually returned errors whenever it successfully accepted a query. An email notifying the

¹⁹ Details here: <https://docs.oracle.com/javase/7/docs/api/javax/swing/JPanel.html>

²⁰ Details here: <https://docs.oracle.com/javase/7/docs/api/javax/swing/JLayeredPane.html>

²¹ Details here: <https://docs.oracle.com/javase/7/docs/api/javax/swing/JComboBox.html>

²² Details here: <https://docs.oracle.com/javase/7/docs/api/javax/swing/JLabel.html>

²³ See here: <http://collection.britishmuseum.org/>

²⁴ See here: <http://americanart.si.edu/collections/search/lod/about/>

²⁵ See here: <http://britishart.yale.edu/collections/using-collections/technology/linked-open-data>

²⁶ See here: <http://data.archaeologydataservice.ac.uk/page/>

²⁷ See here: <http://www.clarosnet.org/XDB/ASP/clarosHome/>

British Museum resulted in a resolution after three weeks; the cause was a technical problem with the Linked Data server.

Problems with the reliability of the servers at an early stage of the project led to concern that the finished application could not depend on SPARQL endpoints being available. This spurred preparation for the use of backup database files held on a user's hard disk but the work was discontinued when the British Museum SPARQL endpoint stabilised.

6.2.2 Inconsistent Implementations of the CIDOC CRM

Inconsistencies in the implementation of the CIDOC CRM standard to collection databases resulted in a significant amount of time being spent on testing and analysing the data. There were two types of inconsistency: data being held in mixed formats that aren't queryable through pure CIDOC CRM queries; and data being held inconsistently with CIDOC CRM best practise.

The Archaeology Data Service holds data in a mixed format: partially in the CIDOC CRM format and partially in the CRM-EH environmental archaeology extension. The CRM-EH extension models processes and concepts used by English Heritage (EH)²⁸ archaeological teams. Accommodating this extension would have required SPARQL queries of increased complexity and was beyond the scope of the project so this service is not supported by the application.

The Yale Center for British Art publishes a CIDOC CRM database but the data is not usable for the purposes of this application. The dates of production of artworks are not linked to the objects so it is not possible to place objects chronologically on a timeline. Dates for objects do exist in the database but it is not possible to link them to the objects through SPARQL queries; this service is therefore not compliant with the application.

The use of different versions of the CIDOC CRM ontology by institutions initially caused confusion due to a lack of documentation. The official CIDOC CRM implementation is used by the Smithsonian American Art Museum and others while a different implementation named Erlangen is used by the British Museum.²⁹ Differences in the names and types of properties and classes between these implementation cause code written to query one implementation failing if used to query the other. This problem was overcome by giving the user control over the URI identifying an implementation for each SPARQL Endpoint Source and by writing more complex SPARQL queries that could accommodate the differences.

²⁸ English Heritage is a charitable trust that cares for over 400 historic sites throughout England. More information can be found at <http://www.english-heritage.org.uk/about-us/>.

²⁹ The response from the CLAROS team about their Linked Data server suggested that they use an even older implementation for their database but the lack of access to a functioning SPARQL endpoint blocked an opportunity to test whether it was compatible with the application.

6.2.3 Writing the SPARQL Queries

The application was written around a single SPARQL query that would return objects from collection databases along with information expected from a CIDOC CRM dataset. There were however three main that caused difficulty in producing a final usable query: the CIDOC CRM is implemented differently within the databases; objects are stored with varying amounts of data; and the British Museum SPARQL endpoint was inconsistent in handling queries.

The CIDOC CRM is implemented in collection databases with different properties used for the same types of information and different paths to the same types of information. For example, the CIDOC CRM allows for a property 'main representation' which is implemented within the Smithsonian American Art Museum collection but is replaced with a custom non-CIDOC-CRM property within the British Museum collection. The SPARQL query building tool within the application had to be written to take such inconsistencies into account.

Details about objects are also variable; this is to be expected with databases of collections hundreds of years old (as in the case of the British Museum). There were also differences between institutions in the types of data they hold about objects in their collections; for example, the Smithsonian American Art Museum doesn't include information about the location of origin of an object whereas the British Museum does. The type of data the user can query through the application is therefore limited to a subset of CIDOC CRM properties (only a small number of which are held by either institution anyway).

The British Museum SPARQL endpoint was very particular about the formatting of queries it would accept through Apache Jena. Queries that the Smithsonian endpoint could accept, or that the British Museum could accept through their SPARQL endpoint online front end, would often fail when sent directly to the SPARQL endpoint from the application.

The British Museum usually failed to accept SPARQL aggregation queries that return all data related to an object as a single row. These aggregation queries were tested and proven to be valid SPARQL queries through an online SPARQL query validation tool.³⁰ With no documentation explaining how these aggregate functions should be written to conform to the British Museum standard, debugging these queries was extraordinarily time-consuming.

A conclusion was reached that the British Museum SPARQL endpoint was reformatting queries pasted through the online front end of the SPARQL endpoint but this was not occurring if they were received directly from the application. Due to time-constraints the use of aggregate functions to extract deduplicated results from the application had to be set aside. The deduplication was instead implemented within the application; duplicate results are returned from queries but filtered through the object's identification codes. This

³⁰ As checked here: <http://sparql.org/query-validator.html>

method is not ideal but fulfils the requirements of the application and also results in faster query running times.

6.2.4 Querying Years and Dimensions

Two dimensions were difficult to query due to a lack of standardization of the data; the year of creation of an object and the dimensions of an object. Querying the year was resolved to a degree but querying by dimension was abandoned for this iteration of the application.

The date of creation of an object was not in a standardized format within the British Museum database; dates are recorded as year numbers, year numbers with the era (AD or BC) in different positions or formatted differently within the field, or as textual descriptions of periods of time (“8th – 9th C”). To simplify the SPARQL queries (to accommodate the inconsistent behaviour of the British Museum SPARQL endpoint) only numerical representations of AD-era years were supported.

Tests were also taken to support filtering results by dimension (length, width and height) and unit of size (mm, cm, m, km). Unfortunately, because the units measuring object dimensions are unknown at query time the only method to do this would be to check dimensions by multiple unit sizes within a single query.

A SPARQL query satisfying this requirement would be overly complex and endpoints would take substantially longer time to process. A conversion method within the application would also have been required to convert units of object dimensions into the units chosen by the user. Given the drawbacks and time-consuming work that would have been required to complete this feature it was decided to not be implemented.

6.2.5 Displaying the Best Division of Results

The specification required that the user would be able to divide the results from a query into groups that could be further divided and that an algorithm would attempt to choose the best dimension to divide the data by. The goal of the algorithm is the best possible distribution of results across the maximum number of categories.

The first stage of the algorithm removes any filter option that would result in only one TimeLine being displayed; this ensures that only filters that will display a breakdown of the chosen TimeLine can be chosen; there would be no significant change to the visualization if only one TimeLine was displayed as the result of an option. Implementing this feature required careful design of the Collection Display Panel and Visualization Tab classes; with filtered results being passed between them in stages matching the drill-down into the data.

The second stage of the algorithm scores dimensions of the data based on a set of objects and chooses to filter by the highest scoring dimension. The Query and Source dimensions have a handicap because they are intrinsically less interesting than other dimensions (due to the number of categories having been defined by the user). The scores for dimensions are calculated dependent on the number of categories within the results and the evenness of distribution of objects across categories. The user can override the automated choice.

Pseudo-code of the second stage of the algorithm is below: the count of appearances of each variable multiplied are multiplied together then multiplied again by the number of variables within the dimension (subject to a limit) to obtain a score. A handicap is then applied to the Query or Source dimensions. The highest scoring dimension is returned as the chosen filter.

```
chooseFilterMethod() {  
    finalChoice  
    maxScore = 0;  
    For each Dimension (Query, Source, Creator, Depicts, etc.)  
        tempScore = 1;  
        For each variable (the breakdown of the Dimension) {  
            tempScore = tempScore * ( count of variables in dimension )  
        End For  
        tempScore = tempScore * min of (variable count or max timelines - 1)  
        If (Dimension = Query or Source) {  
            tempScore = tempScore / 2          // this is the handicap  
        }  
        If (tempScore > maxScore) Then  
            finalChoice = this Dimension  
            maxScore = tempScore  
        End If  
    End For  
    return finalChoice  
}
```

After a dimension is chosen, the display of the TimeLines is ordered by the count of objects within each TimeLine, with larger counts being placed higher on the visualization. This ensures the largest groups are placed most prominently; implementing this required careful design of a Java 8 stream with ordering in reverse.

Dimension	Description
Query	A list of the queries that gathered the objects in the collection
Source	A list of the Sources that the user has queried
Depicts	People, things and concepts displayed in images on objects
Material	The physical makeup of objects
Object Type	A category describing objects according to common nouns for things
Technique	The method by which the object was produced

Table 11: dimensions used to filter the results.

6.2.6 Implementing the TimeLines

Creating a functioning TimeLine visualization required careful manipulation of Swing components and Java graphical drawing methods. Major difficulties included: the placement of Entities and year labels along the TimeLine; and the implementation of zooming into the TimeLines with a sliding scale.

Enabling zooming into and scrolling along the TimeLines required careful manipulation of the ratio of sizes of the TimeLines and the JScrollPane that allows users to navigate the TimeLines. This was resolved by creating a simple model of changes in the ratios of sizes of components in an Excel worksheet followed by experimentation with an open running debug version of the application where code was manipulated so that changes took immediate effect.

7 Testing

The software release life cycle of an application has the software tested in stages before a version is ready for release. The production of this application followed a typical testing and development process to ensure the final version is stable.

7.1 Pre-Alpha Testing

The pre-alpha stage refers to activities performed in the production of the application before a finished version is available for testing. This includes the gathering of specifications, the application design and the implementation of the software described in the preceding chapters. This stage does however include some specific testing activities deserving of mention.

7.1.1 Unit Testing

Unit testing is the subjection of small sections of code such as objects or methods to a series of code tests that are written in specific testing packages. Within the test-driven development programming methodology unit tests are written before the application code to force the developer to work to coded criteria. During the development of the application unit tests are also created in response to the discovery of bugs so that the code can be confirmed as fixed.

Unit tests were applied during the implementation of this application. The test-driven development methodology was used for complex code fragments such as the methods for calculating dates on the TimeLines (within the Collection Display Utilities class). Debugging sessions also resulted in the creation of further unit tests.

The JUnit4 testing framework was used for the creation of the test classes applied to this application.³¹ Other Java testing frameworks are available but JUnit4 was included by default with the Netbeans IDE and there was no reason to switch to another. To avoid overly complicating tests including multiple objects a mock object framework named Mockito was also imported into the testing suites.³²

³¹ Further information about JUnit4 may be found here: <http://junit.org/junit4/>

³² Further information about the Mockito mock framework may be found here: <http://mockito.org/>

7.1.2 Debugging

The Netbeans IDE has a visual debugging tool with two features that assisted the implementation of the project: the ability to take GUI snapshots and to apply changes to the code while the application is running. The GUI snapshot tool enabled quick location of the code for GUI elements through visual inspection of the GUI. The ability to change code while the application was running assisted on occasions when graphical elements did not appear as expected; the rapid visual feedback enabled quick corrections to the code.

7.2 Alpha Testing

Alpha testing of an application takes place when the application is substantially complete and is performed by the members or employees of the development team. The purpose of this stage is to measure an application against benchmarks that describe minimum functionality. The table on the following page lists the benchmarks for this application and the results of the initial tests. Running through the tests uncovered a number of problems, ranging from missing functionality to the cosmetic.

The 'edit' function for the Source Management Tool failed to save correctly due to the code not being updated when the extra field for CIDOC CRM implementations were added to the SPARQLEndpoint class. This was corrected immediately as this is essential functionality for user control of the sources of data.

Testing a query with only a maximum date gave an unexpected result; objects without dates were also returned (they can't be viewed on the visualization but can be viewed on the Entities Tab). The code was modified to ensure only objects with a date would be returned in this case.

The test of a single query through the technique field failing led to a long investigation into the compatibility of the SPARQL query used and the British Museum SPARQL endpoint. The query was confirmed as valid SPARQL code through an online validation tool, being processed correctly by the Smithsonian American Museum of Art SPARQL endpoint, and even by being accepted by the British Museum SPARQL endpoint but *only* if entered into the online webpage fronting the endpoint. The query would only fail if sent to the British Museum SPARQL endpoint directly from the application. Extensive testing was undertaken to create the most robust possible SPARQL query that wouldn't fail when sent to the British Museum SPARQL endpoint.

The problem with the scale on the TimeLine visualization not updating correctly was a problem with consistency; when passing from a view encompassing decades to one encompassing years the scale would sometimes not change and this was dependent on the initial size of the scale. Due to time constraints and being a cosmetic issue, this issue was

given a lower priority to correct and did not delay the release of the application to beta-testing.

The problem with a Collection not saving was a minor issue with the file path having not been changed correctly when the path was changed. This was resolved immediately.

Following the rectification of the issues uncovered through alpha testing the application entered a 'feature freeze' stage. From this point forward no features were added to the application while awaiting feedback from third party beta-testers.

Benchmark	Status
Application	
The application can be opened.	Pass
The application can be closed.	Pass
SPARQL Endpoints	
A SPARQL endpoint can be added to the Source Management Tool	Pass
A SPARQL endpoint can be edited within the Source Management Tool	Fail
A SPARQL endpoint can be deleted from within the Source Management Tool	Pass
Collections	
A Collection can be created	Pass
A Collection can be renamed	Pass
A Collection can be closed	Pass
A Collection can be reopened	Pass
A Collection can be deleted	Pass
Queries	
A single query with no date range is successful	Pass
A single query with a minimum date is successful	Pass
A single query with a maximum date is successful	Unexpected
A single query with minimum & maximum dates is successful	Pass
A single query by name field is successful	Pass
A single query by identifier field is successful	Pass
A single query by depiction field is successful	Pass
A single query by material field is successful	Pass
A single query by type field is successful	Pass
A single query by technique field is successful	Fail
A single query by creator field is successful	Pass
A double query is successful	Pass
A query can be deleted along with the Entities it returned	Pass
Query creation rejects query with no Source entered	Pass
Query creation rejects numbers longer than 4 digits entered into the year fields	Pass
Query creation rejects query with limit over the maximum	Pass

Table 12: Results of the alpha testing, section 1.

Benchmark	Status
Visualization	
First filter correctly filters groups	Pass
Sub-group 1 filters correctly	Pass
Group 1 Filter correctly filters groups	Pass
Sub-group 2 filters correctly	Pass
Group 2 Filter correctly filters groups	Pass
Selecting an Entity updates the Properties window	Pass
Selecting a further Entity updates the Properties window	Pass
The timelines can be zoomed into	Pass
The timelines can be scrolled along while zoomed	Pass
The scale updates successfully on zoom	Fail*
Saving & Loading	
A collection can be saved.	Fail
A collection can be reloaded.	Pass

Table 13: results of the alpha-testing, section 2.

7.3 Profiling

The Netbeans IDE includes a profiling tool that allows a developer to analyse performance of an application while running. Key metrics that were analysed for this application were: CPU usage; JVM garbage collection; memory usage; number of running threads; and number of loaded classes.

The initial test revealed that memory usage expanded rapidly after images were loaded into the system but expanded rapidly and then cycled through rapid spikes of memory usage. Further experimentation revealed that this cycling of heap memory usage only occurred when the Visualization Tab was displayed.

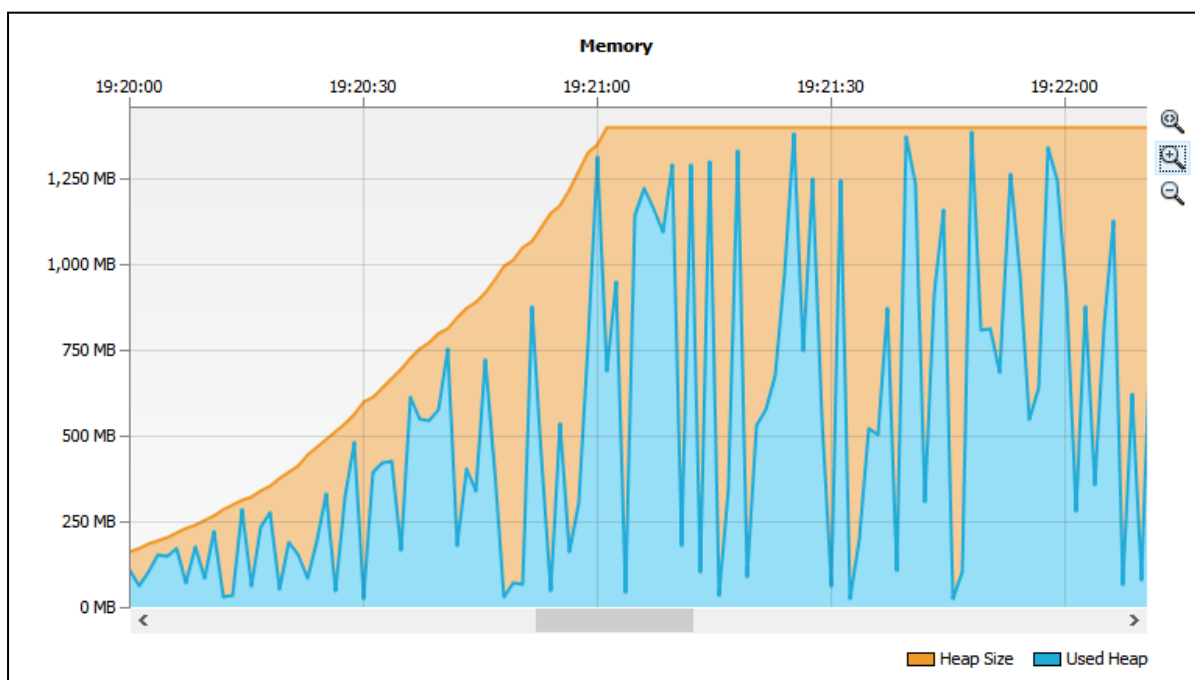


Figure 12: heap memory cycling.

Careful reading of the `CollectionDisplayPanel`, `TimeLine` and `EntitiesDisplay` classes (which were all being repeatedly redrawn when the Visualization Tab was displayed) and placement of debug messages in the code (read through the IDE text output) revealed methods that were running when they should be waiting, and the repeated instantiation of objects that could instead be modified rather than recreated.

Changes were made to the code of the `CollectionDisplayPanel` and `TimeLine` classes that limited the number of method calls and reduced the number of object instantiations. The cycling of heap memory usage was reduced by these changes but not completely removed; time constraints cut short further analysis of this inefficiency.

7.4 Beta Testing

Beta testing involves the release of a substantially complete and tested application to users who check that they can use the application, report any bugs they discover, and possibly suggest additional features that they consider useful. A closed beta includes users invited by the developer while an open beta is a release of the prototype application to an audience who has expressed interest.

7.4.1 Recruiting Testers

Due to restrictions on the time available and the developer's limited relationships with professionals within the cultural heritage sector, an open beta was chosen for the final stage of testing. A search online for organizations of interest to cultural heritage professionals identified the following possible sources of testers:

- CIDOC CRM Documentation Standards Working Group³³
- The Museums Computer Group³⁴
- The UCL Museums & Collections Research Project³⁵
- Curator: The Museum Journal³⁶
- Museums and the Web³⁷
- The British Museum³⁸
- Smithsonian American Art Museum³⁹

These organizations were contacted by email with a short description of the application, an image representing the filtered results of a collection, explanation of the open beta test, and a request for assistance in promoting the test to their staff or members.⁴⁰ A small number of volunteers were recruited through these organizations.

In order to increase the number of potential testers other social media communities with an interest in the subject matter were identified:

- Reddit: Artefact Porn⁴¹
- Reddit: AmericanArt.si.edu Domain⁴²

³³ <http://www.cidoc-crm.org/index.html>

³⁴ <http://museumscomputergroup.org.uk/>

³⁵ <https://www.ucl.ac.uk/museums/research>

³⁶ <http://www.curatorjournal.org/>

³⁷ <http://www.museumsandtheweb.com/>

³⁸ <http://www.britishmuseum.org/>

³⁹ <http://americanart.si.edu/>

⁴⁰ See chapter 11.

⁴¹ <https://www.reddit.com/r/ArtefactPorn/>

⁴² <https://www.reddit.com/domain/americanart.si.edu/>

These communities were contacted through moderators for permission to post a request or via public posts but unfortunately neither returned replies.

The developer also wrote an article for their LinkedIn page outlining the application and calling for beta-testers. This was promoted by a leading member of the UCL Museums & Collections Research Project through their Twitter feed.

In response to these communications, six people volunteered to undertake the beta-testing but only two of the six responded to a later email confirming the date that the testing would begin.

7.4.2 The Testing Process

A user manual was written explaining the use of the application and sent to the beta-testers,⁴³ example Collections were included, and the application was placed as a zip file on a public DropBox account for download.

Unfortunately, delays in completion of the application resulting from problems with the British Museum SPARQL endpoint not accepting valid SPARQL queries left little time for beta-testers to trial the application before the project submission deadline. The number of responders to an email about the new trial date dropped; the momentum which initially spurred volunteers seemed to wane after the delay in making the application available.

Only two volunteers said that they may be able to test the application before the deadline but unfortunately neither replied in time. Delaying the call for beta-testers until software is ready to be distributed would be more likely to convert the volunteers to actual software testers in future; the long lead time seemed to diminish enthusiasm for the product.

⁴³ See the appendices.

8 Summary

The application was completed and fulfils the requirements set out in the proposal; the Source Management Tool, querying tool and the TimeLine visualization were created; TimeLines may be sorted and filtered on multiple dimensions and drilled-down into to assist the user in their analysis of the data; and administrative functions such as saving and loading of the collections, and outputting data to text were also implemented.

8.1 Conclusions

With hindsight, the task of creating a desktop application within the three month timeframe for the project seems ambitious; particularly when the requirements called for complex querying and visualization tools. It could not have been completed without the use of a Rich Client Platform but the complexity of the Netbeans Platform and the steep learning curve delayed progress early in the project and obscured best practises which would have enabled a more idiomatic application of the technology.

Problems with the SPARQL endpoints also hindered progress; when the SPARQL querying tool was initially completed the instability of the endpoints suggested they might never be reliable enough to depend upon for the purpose of demonstrating the application. While communication with the endpoint owners resulted in promises of action, steps were taken to prepare the application for the alternative function of querying backup database files. Ultimately this feature was not required or completed but time was used preparing for the implementation of this function.

The endpoints however are still inconsistent in their ability to accept and process valid SPARQL queries. The British Museum endpoint has a problem accepting queries unless they are sent in a very particular undocumented format; testing these queries was time-consuming and should have been unnecessary. This affected the project in three ways: the use of SPARQL aggregate functions had to be abandoned as even extensive testing couldn't uncover the correct query formatting to use; the open beta-test was significantly delayed; insufficient time was left to improve and debug cosmetic aspects of the TimeLine component.

The different implementations of the CIDOC CRM also caused some difficulty; experimentation with queries and analysis of object data was required before to understand how the CIDOC CRM wasn't being applied consistently. The final application deals with these inconsistencies but the SPARQL queries created by the code are needlessly complex and now support other ontologies which the CIDOC CRM should render redundant. However, the CIDOC CRM is still little implemented and organizations will hopefully learn from these early examples how to structure their data more consistently.

Despite the challenges the application was completed to fulfil the specifications and provides a novel perspective on the cultural heritage data that has emerged on the semantic web. Given the time constraints of this project only core features could be included in the initial specification but there is significant scope for development and improvement of the application.

8.2 Evaluation

Reflection on the project leads the developer to consider that the project was too ambitious for the timeframe and given their experience. Although the program was completed with all components outlined in the specification functioning, some features are underdeveloped and would have benefitted from further examination. Other features, not in the specification but which were intended to improve the user experience, were unfortunately abandoned due to time constraints.

Features of the application that were developed to a high standard were the Source Management Tool, the Properties Pane, saving and loading of Collections to xml, and the core of the TimeLine visualization. The TimeLine visualization gives an intriguing view of the data returned from queries and the object images tempt the user to click on them to see further information in the Properties Pane.

The TimeLine visualization would however have benefitted from further development and it is unfortunate that problems with the SPARQL queries consumed so much development time. Inconsistent changes to the time scale at the bottom of each TimeLine could have been rectified so that the scale always shifts from displaying centuries to decades to years depending on the level of zoom. Difficulty viewing some object images when they are created in the same year and are on the same TimeLines might also have been rectified; there are ways around this, for example, clicking at the edge of a partially obscured object, or changing the breakdown of the TimeLines, but they are not ideal.

The calculation of the breakdown of collections of objects into the TimeLines could benefit from further development to increase the efficiency of the process. The code implementing this feature is complicated and speed improvements could certainly be achieved through refactoring.

It is unfortunate that the SPARQL queries would not consistently work with the British Museum endpoint; the solution would have been neater if aggregate functions were used. The only possible solution to this might be the programmatic construction of the SPARQL queries rather than the current solution that builds the query as a string from constants and methods. This feature would have been too time-consuming to learn within the time

constraints of the project and it is not certain it would solve the problems with the SPARQL query formatting; it should however be tested for a further iteration of the application.

With hindsight, query handling after creation should have been designed differently. Queries can't be edited after creation and there is no ability to view the options that were chosen, the user is dependent on reading the name of the query alone. A query editing tool and a larger table for viewing query parameters would ease the process of writing and organizing queries within a Collection.

Features not in the original requirements but started and not completed (due to difficulties that could not be resolved within the time constraints) were the running of queries in their own threads, display of estimated time for long-running processes, and the querying of local databases as an alternative to the SPARQL endpoints. Having queries run in their own threads would free to user to continue examining other Collections while feedback on long-running processes (the downloading of images and calculation of the breakdown of TimeLines) would assure the user that the program hadn't encountered a difficulty. Querying the local database would have provided a backup for the user when SPARQL endpoints are unstable or unresponsive.

8.3 Further Development

Apart from work to rectify issues identified within the previous evaluation, the most substantial improvement that could be made is increased flexibility for the user to define queries and choose the properties and classes they wish to search for. Providing customization of query parameters would remove the requirement for a single SPARQL query that must account for deviations between CIDOC CRM implementations and would free the user to define parameters based on other ontologies.

This ability to customise the queries would however be at the expense of further complexity for the end user; a problem that this iteration of the application attempted to minimize. A middle path between writing raw SPARQL queries and the simplified GUI used in this application may however be possible; mappings between query terms and properties ontologies could be saved in templates that users attach to Sources. Templates for popular museums could be provided with the application so that inexperienced users can begin immediately while experienced users could create their own mappings and share them.

The visualization of the data could also be improved. Firstly, the user could have more control over the display of the TimeLines: options to modify the size, colour, relative position and number of TimeLines that are displayed would improve the user experience. Secondly, alternative visualizations could be provided: if geographic data was extracted from the databases then objects could be shown on maps at their locations of origin or

discovery; further ahead, if the data is linked to 3D representations of the objects these could be placed within navigable 3D environments to create custom galleries (potentially combined with a VR headset).⁴⁴

⁴⁴ For an example of 3D representations, see here: <https://sketchfab.com/britishmuseum/models>

9 References

1. World Wide Web Consortium: *Semantic Web* [Online]. Available: <https://www.w3.org/standards/semanticweb/>
2. World Wide Web Consortium: *Semantic Web: Linked Data*. [Online] Available: <https://www.w3.org/standards/semanticweb/data>
3. N. Crofts, M. Doerr, T. Gill, S. Stead, M. Stiff (eds.), *Definition of the CIDOC Conceptual Reference Model (Version 5.0.4)* [Online]. Available: http://www.cidoc-crm.org/docs/cidoc_crm_version_5.0.4.pdf
4. M. Doerr and N. Crofts, *Electronic Communication on Diverse Data - The Role of the OO CIDOC Reference Model* [Online]. Available: <http://www.cidoc-crm.org/docs/crmrole.htm>
5. Kalliopi Kontiza, Antonis Bikakis, and Rob Miller, *Cognitive-based Visualization of Semantically Structured Cultural Heritage Data* [Online]. Available: <http://ceur-ws.org/Vol-1456/paper6.pdf>
6. S.K. Card, *Readings in Information Visualization: Using Vision to Think*, Jock D. Mackinlay, Ben Shneiderman, Ed. Morgan Kaufmann Publishers, 1999.
7. D. Rosenberg, A. Grafton, *Cartographies of Time: A History of the Timeline*, Princeton Architectural Press, 2012.

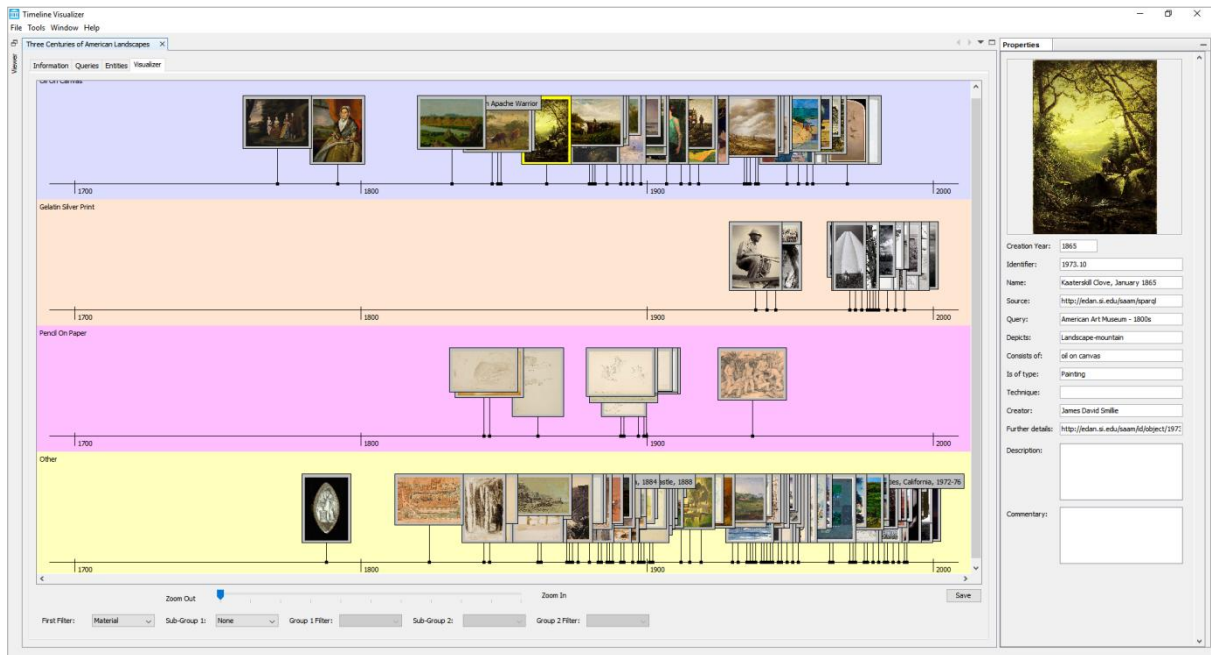
10 Appendix: User Manual

The user manual is on the following pages.

CIDOC CRM

TimeLine Visualizer

User Manual



1 Introduction

This application has been designed to allow users to access cultural heritage collections that have been implemented in accordance with the CIDOC CRM ontology. Users can create simple queries through a user interface. Objects from collections can then be retrieved and viewed either in table or timeline formats.

The initial version is a Windows application. Please email the designer at the email address below if you would like to try a build for an alternative operating system.

[Contact Details](#)

James Hill

jamesghill@yahoo.co.uk

2 Concepts

There are some concepts to familiarise yourself with before proceeding with the instructions for use of the application.

2.1 Sources

Sources are the SPARQL endpoints that access CIDOC CRM implemented databases. There are currently only a small number of these available.

2.2 Queries

These are SPARQL queries created and stored within the application. These Queries retrieve data from Sources.

2.3 Entities

Entities represent objects that are held by museums.

2.4 Collections

Collections hold Queries and Entities plus notes entered by the user in a folder-like package that can be saved and loaded across sessions.

3 Installation

Note: the application requires that Java 7.1 or higher is installed on your machine.

The application will be downloaded as a zip file named 'timelinevisualizer.zip'. Extract the contents to a folder where you would like to store the application. Inside that folder there will be another folder named 'bin'. Within that folder there will be two executable files:

timelinevisualizer.exe

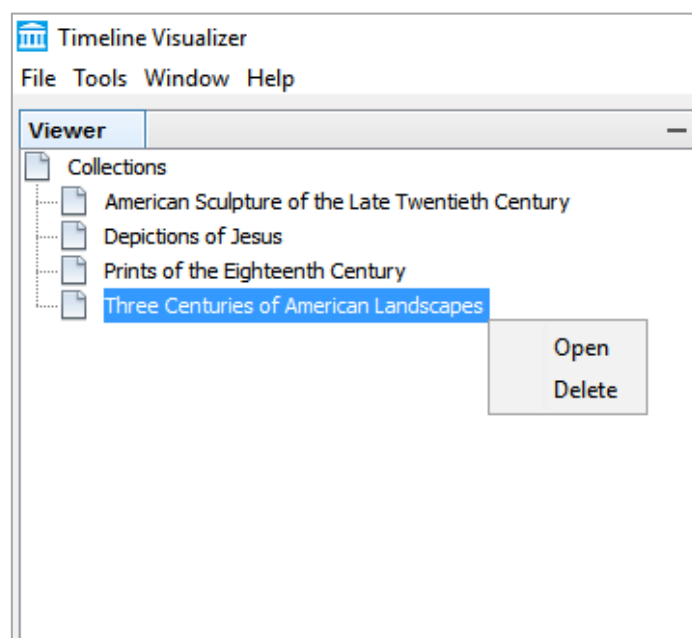
timelinevisualizer64.exe

Double click on either of these two executables to start the application.

4 Getting Started

After installation, open the application through double-clicking one of the executable files you found within the 'bin' folder. The application will load with some Sources and Collections already available.

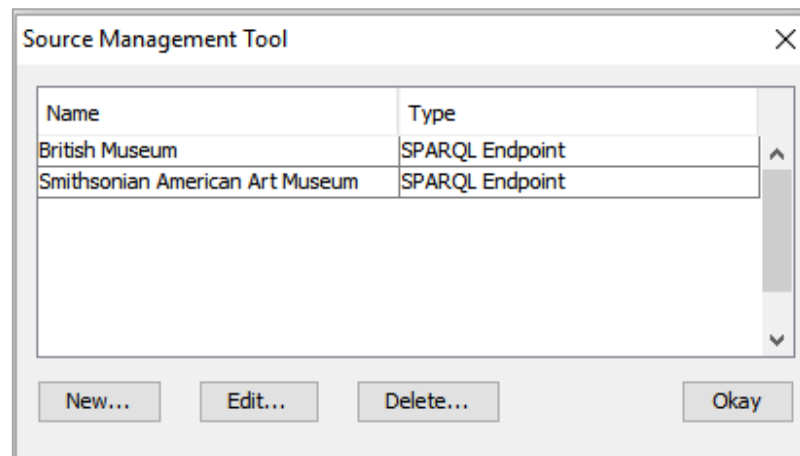
Example Collections are shown in the Viewer pane at the left hand side of the application window. Double-clicking a Collection node will open that Collection. Right-clicking a Collection node gives you the options to either open or delete the Collection. Deleting a collection removes it permanently. Opening an example Collection for the first time will cause the application to start downloading images from the internet so there will be a delay before it appears.



The Sources included with the application are two stable CIDOC CRM databases; the British Museum and Smithsonian American Art Museum SPARQL endpoints. The Sources can be viewed through the Source Management Tool which is accessed through the 'Tools' option in the menu bar. Selecting 'Source Management Tool' will open a window with a table of the existing Sources.

5 Managing Sources

The Management Source Tool is a window with a table displaying the Sources currently loaded into the system. The name given to the Source and the type of Source are displayed. Currently only SPARQL endpoints are supported by the application. The user can interact with the Source Management Tool through three buttons: 'Delete', 'Edit' and 'New'.

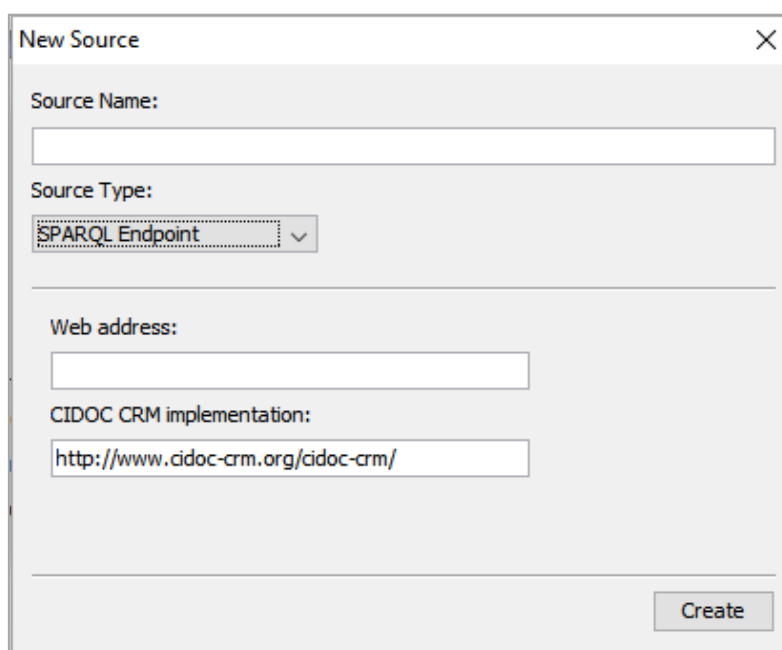


5.1 New

'New' will open up a sub-window where you can define a new Source. There is a text field where you can enter the name of the Source and a box for choosing different Source types; currently only a SPARQL endpoint can be chosen.

After the Source type is selected two further fields become available: the first requires the URL for the SPARQL endpoint which can be found through the websites of museums; the second is the implementation of the CIDOC CRM used to structure the data at the endpoint. The CIDOC CRM default is filled in by default but there are others such as 'Erlangen'. These can be identified by examining the 'PREFIX' of example SPARQL queries museums place on their SPARQL endpoints.

Once all fields are filled the user can select 'Okay' to complete the creation of the Source.

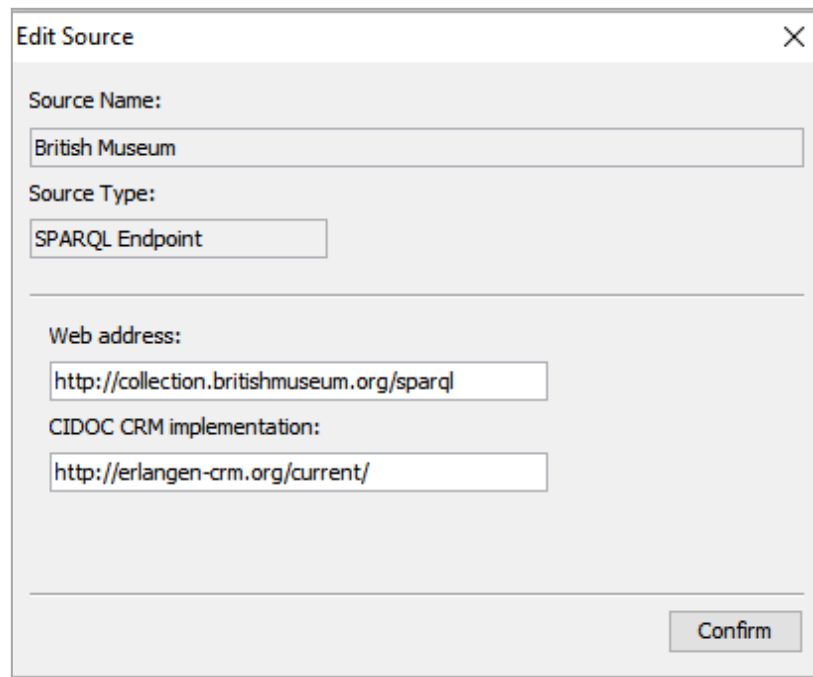


The image shows a 'New Source' dialog box with a close button (X) in the top right corner. The dialog contains the following fields and controls:

- Source Name:** A text input field.
- Source Type:** A dropdown menu with 'SPARQL Endpoint' selected.
- Web address:** A text input field.
- CIDOC CRM implementation:** A text input field containing the URL `http://www.cidoc-crm.org/cidoc-crm/`.
- Create:** A button located at the bottom right of the dialog.

5.2 Edit

Selecting a Source within the table and then 'Edit' opens a window similar to the 'New' Source window. The Source name and type fields are disabled but you may change the SPARQL endpoints URLs and CIDOC CRM implementations.



The screenshot shows a dialog box titled "Edit Source" with a close button (X) in the top right corner. The dialog contains several input fields and a "Confirm" button at the bottom right. The fields are as follows:

- Source Name:** A text box containing "British Museum".
- Source Type:** A dropdown menu showing "SPARQL Endpoint".
- Web address:** A text box containing "http://collection.britishmuseum.org/sparql".
- CIDOC CRM implementation:** A text box containing "http://erlangen-crm.org/current/".

The "Source Name" and "Source Type" fields are disabled, while the "Web address" and "CIDOC CRM implementation" fields are active.

5.3 Delete

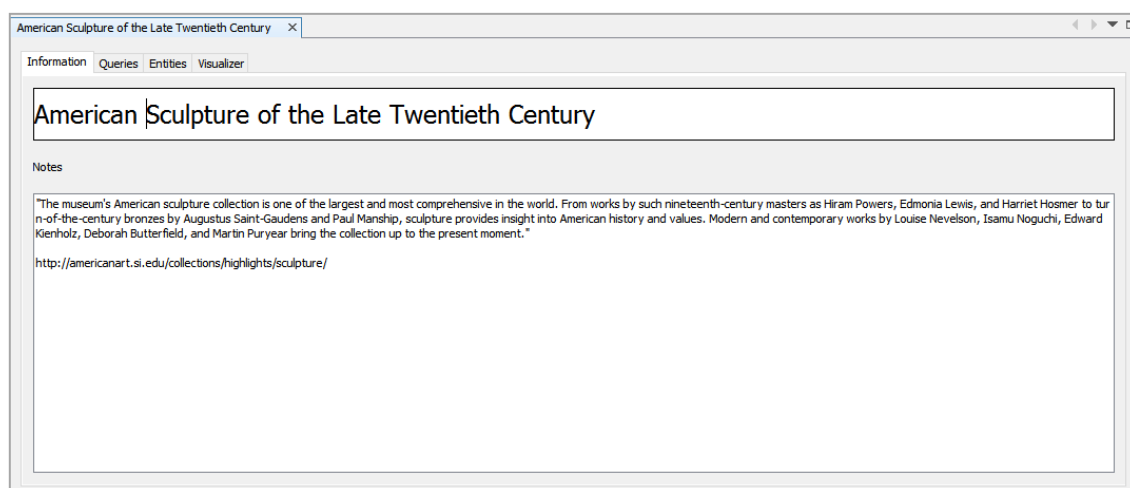
Selecting a Source within the table and then 'Delete' removes that Source from the table and application permanently.

6 Creating Collections

To start creating a Collection you can select 'File' in the menu bar and then 'New Collection'. A Collection window then appears in the centre of the screen. Multiple Collections can be worked on at the same time and will appear as tabs within the same window space. The Collections window has four tabs in which you can work.

6.1 Information

The Information tab has two fields; a Collection name field and a field for containing notes about the Collection. A name based on the date and time will be filled automatically upon creation of a collection. The user can write over that name, press Enter, and the new name will be assigned to the collection throughout the application. Notes can be entered into the Notes field; they will be saved when the Collection is saved.



6.2 Queries

The Queries tab has two sections; a Queries Table on the left and a Query Builder tool on the right. Queries are added to the table when they are created. Selecting a Query and pressing the 'Delete' button the query will permanently remove it and all of the Entities associated with it if it has already been run.

The Query Builder contains options to define your Queries. Your Query must have a name and a Source to be valid. When your Query is complete you may press 'Create' to add the Query to the Queries Table.

The 'limit rows' field will limit the number of rows matching the criteria that the SPARQL Endpoint will check. This count of rows will include many duplicates Entities which will be filtered out by this application. The count of rows should be used to estimate the likely time the Query will take to complete. You should test individual Queries with increasing numbers of rows when you first use the application (for example, 20 rows, then 50 rows, then 100 rows). This will demonstrate how long Queries can take, and how many unique results are likely to be returned, with different numbers of rows.

When all Queries have been defined for a Collection you may press 'Run All' to start the Queries running. You will be warned that the process may take a number of minutes and can either proceed or stop the Queries from running. The application will freeze while the Query is running so it's best to check at this point that you have setup your Queries correctly.

The screenshot shows a web application window titled "American Sculpture of the Late Twentieth Century". The interface has four tabs: "Information", "Queries", "Entities", and "Visualizer". The "Queries" tab is selected.

On the left, under "Existing Queries", there is a table:

Name	Last Run	Query Type
1950s		SPARQL Endpoint
1960s		SPARQL Endpoint
1970s		SPARQL Endpoint
1980s		SPARQL Endpoint
1990s		SPARQL Endpoint

Below the table are "Delete" and "Run All" buttons.

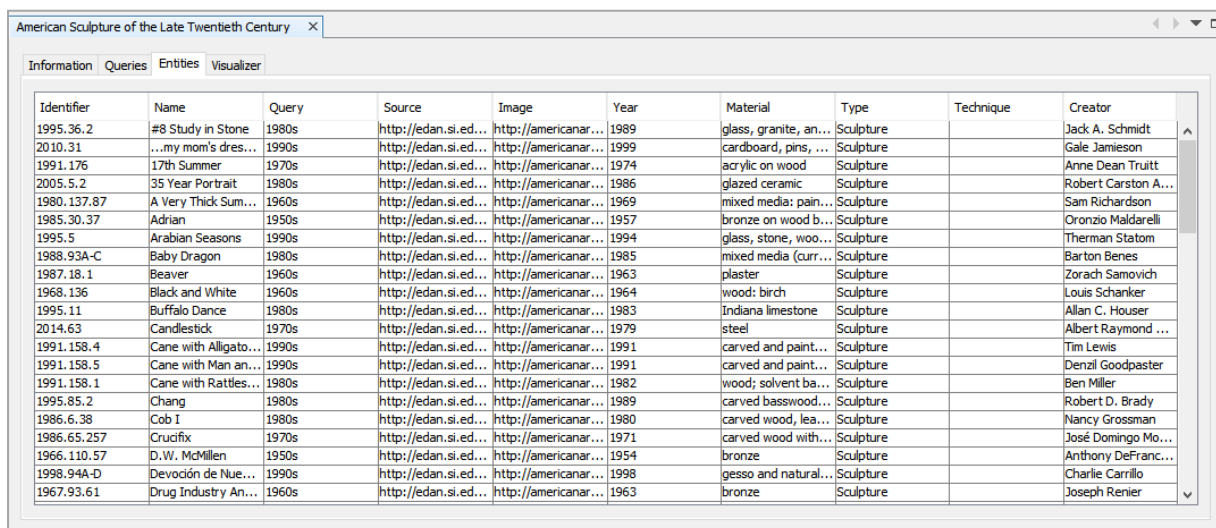
On the right is the "Query Builder" section. It includes:

- A "Name:" text input field.
- A "Source:" dropdown menu with "Select . . ." selected.
- A "Creation year between (AD only):" section with two text input fields and an "and" label.
- Several checkbox options, each with a text input field:
 - ☐ Has name:
 - ☐ Has identifier:
 - ☐ Is a depiction of what?:
 - ☐ Consists of which material?:
 - ☐ Is what type of object?:
 - ☐ Created by technique?:
 - ☐ Created by who?:
 - ☒ Has image?:
- A "Has row limit (maximum = 1000):" section with a checked checkbox and a text input field containing "200".

At the bottom right of the Query Builder are "Reset" and "Create" buttons.

6.3 Entities

The Entities tab displays on a table the Entities that have been returned from previously run Queries within the Collection. This allows you to check the dates in a linear format. The columns can be sorted, resized and moved. Entities without a 'year' will appear in the table even though they won't appear on the TimeLine.



Identifier	Name	Query	Source	Image	Year	Material	Type	Technique	Creator
1995.36.2	#8 Study in Stone	1980s	http://jedan.si.ed...	http://americanar...	1989	glass, granite, an...	Sculpture		Jack A. Schmidt
2010.31	...my mom's dres...	1990s	http://jedan.si.ed...	http://americanar...	1999	cardboard, pins, ...	Sculpture		Gale Jamieson
1991.176	17th Summer	1970s	http://jedan.si.ed...	http://americanar...	1974	acrylic on wood	Sculpture		Anne Dean Truitt
2005.5.2	35 Year Portrait	1980s	http://jedan.si.ed...	http://americanar...	1986	glazed ceramic	Sculpture		Robert Carston A...
1980.137.87	A Very Thick Sum...	1960s	http://jedan.si.ed...	http://americanar...	1969	mixed media: pain...	Sculpture		Sam Richardson
1985.30.37	Adrian	1950s	http://jedan.si.ed...	http://americanar...	1957	bronze on wood b...	Sculpture		Oronzo Maldarelli
1995.5	Arabian Seasons	1990s	http://jedan.si.ed...	http://americanar...	1994	glass, stone, woo...	Sculpture		Therman Statom
1988.93A-C	Baby Dragon	1980s	http://jedan.si.ed...	http://americanar...	1985	mixed media (curr...	Sculpture		Barton Benes
1987.18.1	Beaver	1960s	http://jedan.si.ed...	http://americanar...	1963	plaster	Sculpture		Zorach Samovich
1968.136	Black and White	1960s	http://jedan.si.ed...	http://americanar...	1964	wood: birch	Sculpture		Louis Schanker
1995.11	Buffalo Dance	1980s	http://jedan.si.ed...	http://americanar...	1983	Indiana limestone	Sculpture		Allan C. Houser
2014.63	Candlestick	1970s	http://jedan.si.ed...	http://americanar...	1979	steel	Sculpture		Albert Raymond ...
1991.158.4	Cane with Alligato...	1990s	http://jedan.si.ed...	http://americanar...	1991	carved and paint...	Sculpture		Tim Lewis
1991.158.5	Cane with Man an...	1990s	http://jedan.si.ed...	http://americanar...	1991	carved and paint...	Sculpture		Denzil Goodpaster
1991.158.1	Cane with Rattles...	1980s	http://jedan.si.ed...	http://americanar...	1982	wood; solvent ba...	Sculpture		Ben Miller
1995.85.2	Chang	1980s	http://jedan.si.ed...	http://americanar...	1989	carved basswood...	Sculpture		Robert D. Brady
1986.6.38	Cob I	1980s	http://jedan.si.ed...	http://americanar...	1980	carved wood, lea...	Sculpture		Nancy Grossman
1986.65.257	Crucifix	1970s	http://jedan.si.ed...	http://americanar...	1971	carved wood with...	Sculpture		José Domingo Mo...
1966.110.57	D.W. McMillen	1950s	http://jedan.si.ed...	http://americanar...	1954	bronze	Sculpture		Anthony DeFranc...
1998.94A-D	Devoción de Nue...	1990s	http://jedan.si.ed...	http://americanar...	1998	gesso and natural...	Sculpture		Charlie Carrillo
1967.93.61	Drug Industry An...	1960s	http://jedan.si.ed...	http://americanar...	1963	bronze	Sculpture		Joseph Renier

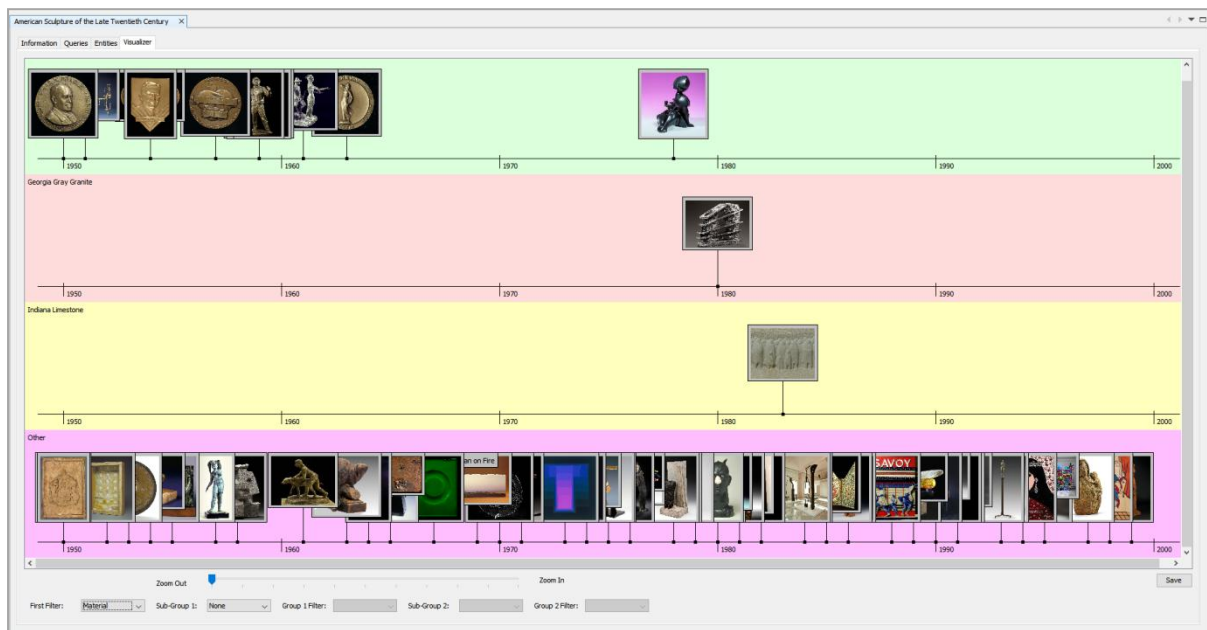
6.4 Visualization

The Visualization tab displays the Entities included within the Collection in a graphical form along TimeLines. TimeLines are themed groups of Entities that are placed along a scale showing years, decades or centuries. These TimeLines divide the collection of Entities into groups that can be selected by the user. Entities are displayed upon the TimeLines as either images of the object (where available) or the name of the object in a banner.

If the mouse is held over an image of an Entity on the TimeLine then the image will be enlarged. Moving the mouse out of range or left-clicking the enlarged image will close it. Left-clicking on the Entity within the TimeLine will pass information about that Entity to the Properties window at the far right of the application.

The filters along the bottom of the screen allow you to drill down into categories of the data. You must work from left to right. Only categories where there is more than one option are displayed. The slider allows you to zoom into the TimeLines when the scale exceeds a length where individual years can be displayed.


If you would like to save the Collection and results to be used in a future session or passed to another user you can save the collection by pressing the 'Save' button at the bottom right of this tab.



7 Properties

When an Entity is selected on a Timeline, information about the object it represents is passed to the Properties panel on the right side of the application. The Properties panel displays an enlarged image of the object along with other information that is available from the database. Information about an Entity remains on the panel until another Entity is selected. The panel can be hidden.

Properties



Creation Year:

1807

Identifier:

PPA81648

Name:

The head of the poll, or the Wimbledon

Source:

<http://collection.britishmuseum.org/spai>

Query:

1800s

Depicts:

Jesus Christ

Consists of:

paper

Is of type:

print

Technique:

Creator:

Charles Williams

Further details:

<http://collection.britishmuseum.org/id/o>

Description:

(For description see other impression).
May 1807

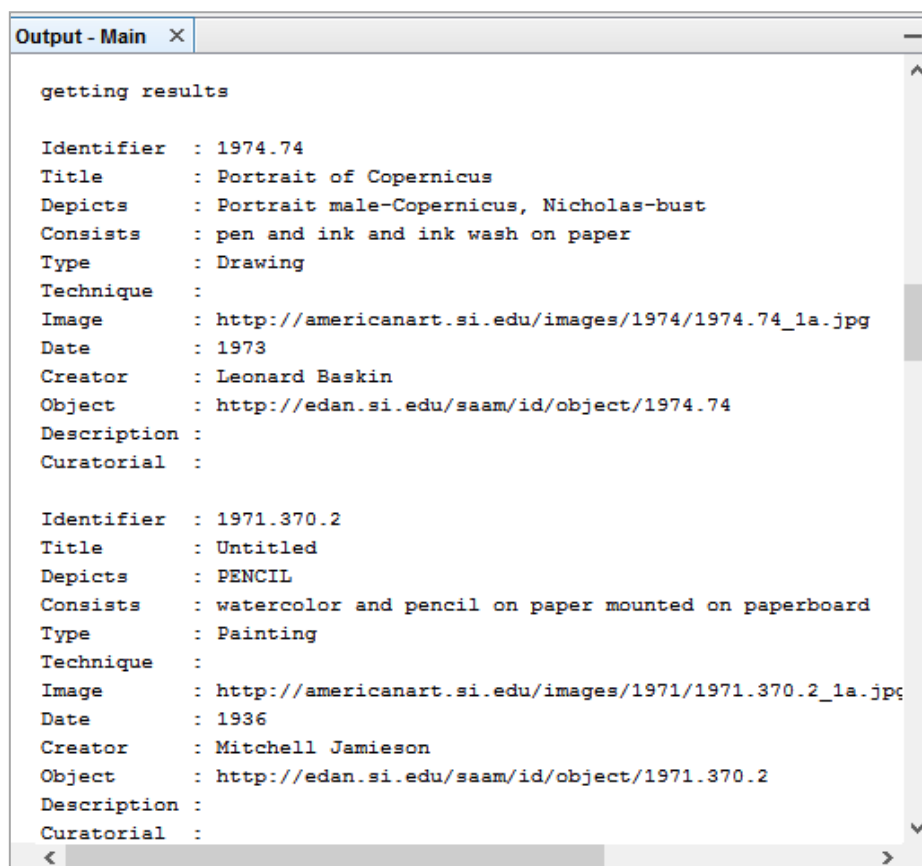
Hand-coloured etching

Commentary:

NB The register number stamped on the print is a duplicate (the number also belongs to a satirical print). I have therefore added an asterisk to the number.

8 Output

Selecting the 'Windows' option on the menu bar reveals an option for 'Output'. Selecting this option reveals a text output window at the bottom left of the screen. Information about actions that are taking place and details about the queries can be found in this window.



9 Uninstallation

To uninstall, delete the folder you extracted the contents of the zip file named 'timelinevisualizer.zip' into on installation.

10 Tips

Check that you are likely to return results before you set up a complex set of Queries: run a number of Queries checking your variables individually with very a small count of rows to inspect (20 or less). If Entities are returned by these smaller Queries you can be more certain that a more complex Query will return results.

If you have problems with any of the windows in the application disappearing or appearing in incorrect positions upon restarting the application then follow these steps: firstly, close down the application if it is open; secondly, navigate your drives to find the folder `C:\Users\{ YOUR USER NAME HERE }\AppData\Roaming`; thirdly, delete the folder `‘.timelinevisualizer’` within the `‘Roaming’` folder. The application windows should start from their usual positions when you restart.

11 Appendix: Recruiting Testers

11.1 Email Description of the Application

On the following page is a copy of the email that was sent out to various organizations in order to recruit beta testers.

The Cultural Heritage Timeline Visualizer is an application that allows users to query online museum collections and view the returned objects on a series of timelines that can be filtered by category. For example, a user may search the British Museum collection for 'painting' between '1900' and '1920' and view the results with the ability to drill-down into the different categories of the paintings. Selecting the images that represent objects will present further details of the objects in a properties window.

The use is able to collect objects from multiple queries at the same time, including queries sent to different online museum collections. The final version will allow the users own collections of objects to be saved for future reference and offline analysis.

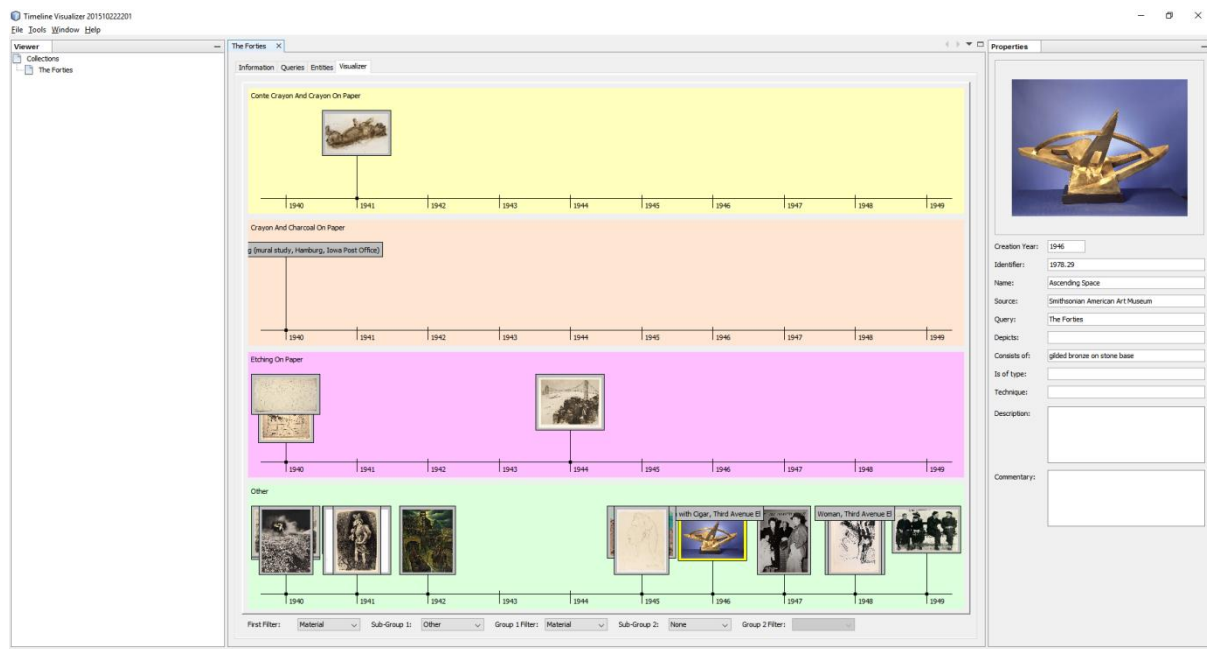
The application is a simple prototype that has been created for an MSc Computer Science project at Birkbeck, University of London, but may be released with extra features in future. The designer is opening up an early version for beta testing. More information about the designer can be found on LinkedIn:

<https://uk.linkedin.com/in/jamesgeoffreyhill>

Please feel free to pass this email on to anyone who you think may be interested in testing the application. If you would like to be a beta tester or have further questions please email:

jamesghill@yahoo.co.uk

The following image gives an impression of the application:



12 Appendix: Code

The following pages contain samples of the code for the Java classes created for this project; all of the code written can be found on the accompanying USB stick. The code has been chosen for demonstrating the classes with the least boilerplate code.

A Netbeans Platform application also depends on the base modules which provide the framework around which a program is built; these are not included on the following pages.

12.1 Collections Module

This section contains all Java code from the Collections module apart from the `CollectionDisplayPanelBeanInfo.java` and `TimeLineBeanInfo.java` classes because they are almost entirely auto-generated code.

12.1.1 CollectionImpl.java

```
package org.jghill.timelinesvisualizercollections;

import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;
import org.jghill.timelinesvisualizercollections.container.CollectionContainer;
import org.jghill.timelinesvisualizercollections.gui.CollectionTopComponent;
import org.jghill.timelinevisualizerentitiescollection.EntitiesCollection;
import org.jghill.timelinevisualizerqueriescollection.QueriesCollection;
import org.openide.util.Lookup;
import org.openide.util.lookup.AbstractLookup;
import org.openide.util.lookup.InstanceContent;

/**
 * A project for holding data from a set of queries.
 *
 * @author JGHill
 */
public class CollectionImpl implements Collection,
Comparable<CollectionImpl>, Lookup.Provider {

    private String name;
    private String notes;

    private final Lookup lu;
    private final InstanceContent instanceContent;
    private EntitiesCollection entities;
    private final QueriesCollection queries;
    private CollectionTopComponent tc;
    private final PropertyChangeSupport pcs = new
PropertyChangeSupport(this);

    /**
```

```

* A constructor accepting containers.
*
* @param name a name for the project.
* @param entities an entities container.
* @param queries a queries container.
*/
public CollectionImpl(
    String name,
    EntitiesCollection entities,
    QueriesCollection queries
) {

    this.name = name;
    this.entities = entities;
    this.queries = queries;
    instanceContent = new InstanceContent();
    lu = new AbstractLookup(instanceContent);

    instanceContent.add(
        new CanOpen() {
            @Override
            public void open(Collection coll) {
                if (tc == null || !tc.isOpened()) {
                    CollectionContainer.addToLookup(coll);
                    CollectionTopComponent collTC = new
CollectionTopComponent();
                    collTC.open();
                    collTC.requestActive();
                }
            }
        }
    );

    instanceContent.add(
        new CanDelete() {
            @Override
            public void delete(Collection coll) {
                CollectionContainer.deleteCollection(coll);
                if (tc != null) {
                    tc.close();
                    tc = null;
                }
            }
        }
    );

}

@Override
public String getName() {
    return name;
}

@Override
public void setName(String name) {
    String oldName = this.name;
    this.name = name;
    pcs.firePropertyChange("name", oldName, name);
}

@Override

```

```

    public String getNotes() {
        return notes;
    }

    @Override
    public void setNotes(String notes) {
        this.notes = notes;
    }

    @Override
    public EntitiesCollection getEntitiesCollection() {
        return entities;
    }

    @Override
    public QueriesCollection getQueriesCollection() {
        return queries;
    }

    @Override
    public int compareTo(CollectionImpl o) {
        return name.compareTo(o.getName());
    }

    @Override
    public Lookup getLookup() {
        return lu;
    }

    @Override
    public void setTopComponent(CollectionTopComponent tc) {
        this.tc = tc;
    }

    @Override
    public void clearEntitiesCollection() {
        entities = new EntitiesCollection("Collection");
    }

    @Override
    public void addPropertyChangeListener(PropertyChangeListener listener)
{
        pcs.addPropertyChangeListener("name", listener);
    }

    @Override
    public void removePropertyChangeListener(PropertyChangeListener
listener) {
        pcs.removePropertyChangeListener("name", listener);
    }
}

```


12.1.2 CollectionContainer.java

```
package org.jghill.timelinesvisualizercollections.container;

import java.io.File;
import java.util.Collections;
import java.util.SortedSet;
import java.util.TreeSet;
import org.jghill.timelinesvisualizercollections.Collection;
import org.jghill.timelinesvisualizercollections.node.CollectionChildren;
import org.jghill.timelinevisualizerentities.Entities;
import org.jghill.timelinevisualizerentities.PhysicalThing;
import org.openide.util.Lookup;
import org.openide.util.lookup.AbstractLookup;
import org.openide.util.lookup.InstanceContent;

/**
 * A collection for holding projects.
 *
 * @author JGHill
 */
public class CollectionContainer {

    private static final CollectionChildren CHILDREN = new
CollectionChildren();
    private static final CollectionContainer CONTAINER = new
CollectionContainer();
    private static final SortedSet<Collection> COLLECTION = new TreeSet<>();

    private static final InstanceContent CONTENT = new InstanceContent();
    private static final Lookup LKP = new AbstractLookup(CONTENT);

    private CollectionContainer() {}

    /**
     * Returns the single instance of this singleton pattern.
     *
     * @return the single instance of this container.
     */
    public static CollectionContainer getInstance() {
        return CONTAINER;
    }

    /**
     * A method for adding collections to the container.
     *
     * @param coll the project to be added.
     */
    public static void addCollection(Collection coll) {
        COLLECTION.add(coll);
        CHILDREN.update(COLLECTION.toArray(new Collection[0]));
    }

    /**
     * A method for removing collections from the container.
     *
     * @param coll the project to be added.
     */
    public static void deleteCollection(Collection coll) {
        COLLECTION.remove(coll);
    }
}
```

```

        CHILDREN.update(COLLECTION.toArray(new Collection[0]));
        for (Entities entity :
coll.getEntitiesCollection().getCollectionSet()) {
            if (entity instanceof PhysicalThing) {
                PhysicalThing pt = (PhysicalThing) entity;
                if (pt.getImageFile() != null) {
                    pt.getImageFile().delete();
                }
            }
        }
        File file = new File("Data/Collections/" + coll.getName() + ".xml");
        file.delete();
    }

    /**
     * Returns the size of the collection.
     *
     * @return the size of the collection.
     */
    public static int getSize() {
        return COLLECTION.size();
    }

    /**
     * Returns all the CollectionChildren.
     *
     * @return the CollectionChildren.
     */
    public static CollectionChildren getChildren() {
        return CHILDREN;
    }

    /**
     * Returns the Lookup for this container.
     *
     * @return the Lookup.
     */
    public static Lookup getLookup() {
        return LKP;
    }

    /**
     * Adds a Collection to the lookup for the container.
     *
     * @param coll to be added.
     */
    public static void addToLookup(Collection coll) {
        CONTENT.set(Collections.EMPTY_LIST, null);
        CONTENT.add(coll);
    }
}

```

12.1.3 CollectionDisplayPanel.java

```
package org.jghill.timelinesvisualizercollections.display;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Point;
import javax.swing.JPanel;
import org.jghill.timelinevisualizerentities.ManMadeObject;
import javax.swing.JSlider;
import javax.swing.JViewport;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

/**
 * A Panel for displaying the results.
 *
 * @author JGHill
 */
public class CollectionDisplayPanel extends JPanel implements
ChangeListener {

    private final static int MAX_SIZE = 10000;
    private final static int TIMELINE_HEIGHT = 200;

    private boolean level1 = false;
    private boolean level2 = true;
    private boolean level3 = true;

    private int size;

    private JSlider zoom;

    private ManMadeObject[] collection;
    private TimeLine[] timeLines;
    private int[] dataArray;

    private int dateDifference;

    private ScaleBuilder scaleBuilder;

    private JViewport viewer;

    /**
     * The constructor.
     */
    public CollectionDisplayPanel() {
        setUp();
    }

    /**
     * Sets the initial settings when constructed.
     */
    private void setUp() {
        setLayout(null);
        scaleBuilder = new ScaleBuilder();
    }

    /**
     * Sets the slider for this object.
     */
}
```

```

*
* @param zoom the slider.
*/
public void setSlider(JSlider zoom) {
    this.zoom = zoom;
    this.zoom.addChangeListener(this);
}

/**
 * Sets the Collection that this panel will display.
 *
 * @param timeLinesCollection that will be displayed.
 * @param filter the string to filter by.
 */
public void setArray(TimeLineCollection timeLinesCollection) {
    clear();
    this.collection = timeLinesCollection.getAllTimeLineObjects();
    this.timeLines = timeLinesCollection.getTimeLines();
    viewer = (JViewport) this.getParent();
    size = viewer.getSize().width;
    zoom.setValue(0);
    dateArray = scaleBuilder.createScaleInfo(collection, 1);
    dateDifference = dateArray[dateArray.length-1] - dateArray[0];
    for (TimeLine tm : timeLines) {
        this.add(tm);
    }
    revalidate();
    repaint();
}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    if (timeLines != null) {
        if (size < viewer.getSize().width) {
            size = viewer.getSize().width;
        }
        modifyScaleZoom();
        paintTimeLines();
        changeSize();
    }
    revalidate();
}

/**
 * Places the TimeLines onto the panel.
 */
protected void paintTimeLines() {
    int tlCount = 0;
    for (TimeLine tm : timeLines) {
        tm.setBounds(
            0,
            TIMELINE_HEIGHT * tlCount,
            size,
            TIMELINE_HEIGHT
        );
        tlCount++;
    }
    setUpdate();
}

```

```

/**
 * Check the zoom level and modify the scale if necessary.
 */
private void modifyScaleZoom() {
    int scale = size / viewer.getSize().width;
    if (scale <= 4 && !level1) {
        level1 = true;
        level2 = false;
        level3 = false;
        setUpdate();
        dateArray = scaleBuilder.createScaleInfo(collection, 1);
    } else if (scale > 4 && scale <= 40 && !level2) {
        level1 = false;
        level2 = true;
        level3 = false;
        setUpdate();
        dateArray = scaleBuilder.createScaleInfo(collection, 10);
    } else if (scale > 40 && !level3) {
        level1 = false;
        level2 = false;
        level3 = true;
        setUpdate();
        dateArray = scaleBuilder.createScaleInfo(collection, 100);
    }
}

/**
 * Returns the status of the 'update' variable.
 */
private void setUpdate() {
    for (TimeLine tm : timeLines) {
        tm.setUpdate();
    }
}

/**
 * Returns the array of dates associated with this display.
 *
 * @return the dateArray.
 */
public int[] getDateArray() {
    return dateArray;
}

/**
 * Clears this panel.
 */
public void clear() {
    collection = null;
    dateArray = null;
    removeAll();
    timeLines = null;
    revalidate();
    repaint();
}

private JSlider source;
int lastSourceValue;

/**
 * Update the scaleZoom level when reacting to the slider changing.

```

```

    *
    * @param e the event that's changed.
    */
    @Override
    public void stateChanged(ChangeEvent e) {

        source = (JSlider) e.getSource();
        int sourceValue = source.getValue();

        if (source.getValueIsAdjusting() && sourceValue != lastSourceValue
        && timeLines != null) {
            lastSourceValue = sourceValue;

            int interval = (getDateArray()[getDateArray().length-1] -
            getDateArray()[0]);
            int viewerWidth = viewer.getSize().width;
            int scaleZoom = (int) ((int) viewerWidth + (sourceValue *
            (viewerWidth * ((double) interval / 100))));

            if (scaleZoom >= viewerWidth && scaleZoom <= MAX_SIZE +
            viewerWidth && dateDifference > 10) {

                int position = viewer.getViewPosition().x;
                int halfWidth = viewerWidth / 2;
                int oldCentre = position + halfWidth;
                double ratio = (double) scaleZoom / size;
                int newX = Math.max(0, (int) (Math.round(oldCentre * ratio)
                - halfWidth));

                size = scaleZoom;
                changeSize();

                viewer.setViewPosition(
                    new Point(
                        newX,
                        viewer.getViewPosition().y
                    )
                );

                repaint();

            }

        }

    }

    /**
     * Changes the size of this panel.
     */
    private void changeSize() {
        this.setPreferredSize(
            new Dimension(
                size,
                timeLines.length * TIMELINE_HEIGHT
            )
        );
    }
}

```

12.1.4 EntityDisplay.java

```
package org.jghill.timelinesvisualizercollections.display;

import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Graphics;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.image.BufferedImage;
import java.net.MalformedURLException;
import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JPanel;
import org.jghill.timelinesvisualizerentities.Entities;
import org.jghill.timelinesvisualizerentities.PhysicalThing;
import org.openide.util.Exceptions;
import org.openide.util.Lookup;
import org.openide.util.Utilities;
import org.openide.util.lookup.Lookups;

/**
 * A component for displaying entities.
 *
 * @author JGHill
 */
public class EntityDisplay extends JPanel implements
Comparable<EntityDisplay>, MouseListener, FocusListener, Lookup.Provider {

    private static final int THUMB_SIZE = 100;
    private static final int BOUNDARY = 5;

    private Entities entity;
    private BufferedImage thumb;
    private Lookup lookup;

    private int w = 110;
    private int h = 110;

    /**
     * Constructor.
     */
    public EntityDisplay() {}

    /**
     * Adds an Entities object to this display.
     *
     * @param entity the Entities object.
     */
    public void setEntity(Entities entity) {
        this.entity = entity;
        setUpDisplay();
        addMouseListener(this);
        addFocusListener(this);
    }

    /**
```

```

    * Initial setup of the EntityDisplay.
    */
private void setUpDisplay() {
    lookups = Lookups.singleton(entity);

    this.setLayout(new FlowLayout());
    this.setOpaque(true);
    this.setBackground(Color.LIGHT_GRAY);
    this.setBorder(BorderFactory.createLineBorder(Color.BLACK));

    getImage();

    if (thumb != null) {
        w = thumb.getWidth() + (BOUNDARY * 2);
        h = thumb.getHeight() + (BOUNDARY * 2);
        JLabel picLabel = new JLabel(new ImageIcon(thumb));
        this.add(picLabel);
        PhysicalThing pt = (PhysicalThing) entity;
        try {this.setToolTipText("<html><img src=\"\" +
Utilities.toURI(pt.getImageFile()).toURL() +\">");
        } catch (MalformedURLException ex) {
            Exceptions.printStackTrace(ex);
        }
    } else {
        JLabel label = new JLabel(entity.getName());
        w = label.getPreferredSize().width + (BOUNDARY * 2);
        h = label.getPreferredSize().height + (BOUNDARY * 2);
        this.add(label);
    }
    this.setSize(w, h);
}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
}

/**
 * Requests the image from the entity that this display represents &
 * calculates dimensions.
 */
private void getImage() {
    PhysicalThing pt = (PhysicalThing) entity;
    thumb = pt.getThumb(THUMB_SIZE);
}

/**
 * @return the year from the entity that this display represents.
 */
public Integer getYear() {
    PhysicalThing pt = (PhysicalThing) entity;
    return pt.getTimeSpan();
}

@Override
public int getWidth() {
    return w;
}

@Override
public int getHeight() {

```



```

        return h;
    }

    @Override
    public int compareTo(EntityDisplay o) {
        if (this.getYear() == null || o.getYear() == null) {
            return 0;
        } else {
            return Integer.compare(this.getYear(), o.getYear());
        }
    }

    @Override
    public void mouseClicked(MouseEvent e) {}

    @Override
    public void mousePressed(MouseEvent e) {
        requestFocus();
    }

    @Override
    public void mouseReleased(MouseEvent e) {}

    @Override
    public void mouseEntered(MouseEvent e) {}

    @Override
    public void mouseExited(MouseEvent e) {}

    @Override
    public void focusGained(FocusEvent e) {
        this.setBackground(Color.YELLOW);
    }

    @Override
    public void focusLost(FocusEvent e) {
        this.setBackground(Color.LIGHT_GRAY);
    }

    @Override
    public Lookup getLookup() {
        return lookup;
    }
}

```

12.1.5 ScaleBuilder.java

```
package org.jghill.timelinesvisualizercollections.display;

import java.util.Calendar;
import java.util.GregorianCalendar;
import org.jghill.timelinevisualizerentities.ManMadeObject;

/**
 * Creates the dimensions for the scale that appears within the TimeLines.
 *
 * @author JGHill
 */
public class ScaleBuilder {

    private int zoom;

    /**
     * Constructor.
     */
    public ScaleBuilder() {}

    /**
     * Creates the info needed for the TimeLine.
     *
     * @param collection of objects to calculate from.
     * @return an array of dates for the scales.
     */
    public int[] createScaleInfo(
        ManMadeObject[] collection,
        int zoom
    ) {
        this.zoom = zoom;
        Calendar earliest = calculateEarliest(collection);
        Calendar latest = calculateLatest(collection);
        int interval = calculateInterval(
            earliest,
            latest
        );
        Calendar start = getStart(earliest, interval);
        Calendar end = getEnd(latest, interval);
        int intervalsCount = countIntervals(start, end, interval);
        return getArrayOfDates(start, interval, intervalsCount);
    }

    /**
     * Calculate the earliest date to display on the TimeLine.
     *
     * @param collection the collection of objects to calculate from.
     */
    private Calendar calculateEarliest(ManMadeObject[] collection) {
        Calendar earliest = null;
        Calendar temp;
        for(ManMadeObject e : collection) {
            Integer year = e.getTimeSpan();
            if (year != null) {
                temp = new GregorianCalendar(year, 1, 1);
                if (earliest == null) {
                    earliest = temp;
                } else if (earliest.after(temp)) {

```

```

        earliest = temp;
    }
}

return earliest;
}

/**
 * Calculate the latest date to display on the TimeLine.
 *
 * @param collection the collection of objects to calculate from.
 */
private Calendar calculateLatest(ManMadeObject[] collection) {
    Calendar latest = null;
    Calendar temp;
    for(ManMadeObject e : collection) {
        Integer year = e.getTimeSpan();
        if (year != null) {
            temp = new GregorianCalendar(year, 1, 1);
            if (latest == null) {
                latest = temp;
            } else if (latest.before(temp)) {
                latest = temp;
            }
        }
    }
    return latest;
}

/**
 * Calculates the intervals size.
 *
 * @return the size of the interval.
 */
private int calculateInterval(
    Calendar start,
    Calendar end
) {
    int difference = end.get(Calendar.YEAR) - start.get(Calendar.YEAR);
    if (difference < 10) {
        return 1;
    } else {
        return (int) (Math.pow(10,
Math.ceil(Math.log10(difference/10))));
    }
}

/**
 * Returns the start date for the scale.
 *
 * @param earliest date.
 * @param interval the interval.
 * @return the first year date for the scale.
 */
private Calendar getStart(
    Calendar earliest,
    int interval
) {
    int year = earliest.get(Calendar.YEAR);
    if (interval == 1) {
        return new GregorianCalendar(year - 1, 1, 1);
    }
}

```

```

        } else {
            year = (int) (interval * (Math.floor((double) year /
interval)));
            return new GregorianCalendar(year, 1, 1);
        }
    }

/**
 * Returns the end date for the scale.
 *
 * @param latest date.
 * @param interval the interval.
 * @return the last year date for the scale.
 */
private Calendar getEnd(
    Calendar latest,
    int interval
) {
    int year = latest.get(Calendar.YEAR);
    if (interval == 1) {
        return new GregorianCalendar(year + 1, 1, 1);
    } else {
        year = (int) (interval * (Math.ceil((double) year / interval)));
        return new GregorianCalendar(year, 1, 1);
    }
}

/**
 * Calculates the number of intervals to display on the TimeLine.
 *
 * @return the number of intervals.
 */
private int countIntervals(
    Calendar earliest,
    Calendar latest,
    int interval
) {
    int difference = latest.get(Calendar.YEAR) -
earliest.get(Calendar.YEAR);
    if (interval == 0) {
        return 10;
    } else {
        return ((difference / interval) * zoom) + 1;
    }
}

/**
 * Returns an array of dates from the start date.
 *
 * @param earliest the earliest Calendar date.
 * @param interval the size of the intervals.
 * @param intervalsCount the number of intervals.
 * @return an array of years.
 */
private int[] getArrayOfDates(
    Calendar start,
    int interval,
    int intervalsCount
) {
    int[] dates = new int[intervalsCount];
    if (interval == 0) {

```

```

        dates[0] = 0;
        return dates;
    } else {
        for(int i = 0; i < intervalsCount; i++) {
            dates[i] = start.get(Calendar.YEAR) + ((i * interval) /
zoom);
        }
        return dates;
    }
}
}

```

12.1.6 TimeLine.java

```
package org.jghill.timelinesvisualizercollections.display;

import java.awt.Color;
import java.awt.Graphics;
import java.util.Arrays;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JViewport;
import org.apache.commons.lang3.text.WordUtils;
import org.jghill.timelinesvisualizercollections.gui.CollectionTopComponent;
import org.jghill.timelinevisualizerentities.ManMadeObject;

/**
 * Displays the TimeLine relevant to the selection.
 *
 * @author JGHill
 */
public class TimeLine extends JLayeredPane {

    private final static int DIAMETER = 6;
    private final static int RADIUS = DIAMETER / 2;
    private final static int IMAGE_UPPER = 140;
    private final static int LINE_INDENT = 20;
    private final static int SCALE_INDENT = 40;
    private final static int UPNOTCH = 5;
    private final static int DOWNNOTCH = 15;
    private final static int SCALE_OFFSET_FROM_BOTTOM = 25;
    private static final int LABEL_OFFSET = 5;
    private static final int LABEL_LENGTH = 50;
    private static final int LABEL_HEIGHT = 15;

    private final static int DESCRIPTION_INDENT = 5;
    private final static int DESCRIPTION_LENGTH = 400;
    private final static int DESCRIPTION_HEIGHT = 15;

    private final String name;

    private int[] intervals;
    private final ManMadeObject[] objects;
    private final JLabel description = new JLabel();
    private JLabel[] labels;
    private EntityDisplay[] eDisplays;

    private final CollectionDisplayPanel cdp;

    private int vertical;
    private int lineLength;
    private int scaleLength;

    private final Color color;

    /**
     * Constructor for the TimeLine.
     *
     * @param name to appear on the TimeLine.
     * @param objects to represent on the TimeLine.
     * @param color of the TimeLine background.
     */
}
```

```

public TimeLine(
    String name,
    ManMadeObject[] objects,
    Color color,
    CollectionDisplayPanel cdp
) {
    this.name = name;
    this.objects = objects;
    this.color = color;
    this.cdp = cdp;
    setUp();
}

/**
 * Sets the arrays used to create this TimeLine.
 */
private void setUp() {

    this.setLayout(null);
    this.setOpaque(true);
    this.setBackground(color);

    viewport = (JViewport) cdp.getParent();

    this.add(description);
    description.setVisible(true);
    description.setText(WordUtils.capitalize(name));

    eDisplays = new EntityDisplay[objects.length];
    for(int i = 0; i < eDisplays.length; i++) {
        eDisplays[i] = new EntityDisplay();
        eDisplays[i].setEntity(objects[i]);
        eDisplays[i].addFocusListener(
            (CollectionTopComponent)
cdp.getParent().getParent().getParent().getParent().getParent());
    }

    Arrays.sort(eDisplays);
    for (int i = 0; i < eDisplays.length; i++) {
        if (eDisplays[i].getYear() != null) {
            this.add(eDisplays[i]);
            setComponentZOrder(eDisplays[i], i);
        }
    }
}

private boolean update = true;

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    if (update) {
        setLabels();
    }
    paintTimeLine(g);
}

/**
 * Creates the list of labels to place onto the TimeLine.
 */

```

```

private void setLabels() {
    if (labels != null) {
        for(int i = 0; i < intervals.length; i++) {
            this.remove(labels[i]);
        }
    }
    intervals = cdp.getDateArray();
    labels = new JLabel[intervals.length];
    for(int i = 0; i < intervals.length; i++) {
        labels[i] = new JLabel();
        this.add(labels[i]);
    }
    update = false;
}

/**
 * Paints the scale onto the TimeLine.
 *
 * @param g the Graphics object.
 */
private void paintTimeLine(Graphics g) {

    int height = getHeight();
    int width = getWidth();

    vertical = height - SCALE_OFFSET_FROM_BOTTOM;
    lineLength = width - (LINE_INDENT * 2);
    scaleLength = lineLength - (SCALE_INDENT * 2);

    paintDescription();
    paintScale(g);
    paintEntities(g);
    repaint();

}

private JViewport viewPort;
private int viewPortLastX = -1;

/**
 * Places the description onto the TimeLine.
 */
private void paintDescription() {
    int viewPortX = viewPort.getViewPosition().x;
    if (viewPortX != viewPortLastX) {
        viewPortLastX = viewPortX;
        description.setBounds(
            viewPortX + DESCRIPTION_INDENT,
            DESCRIPTION_INDENT,
            DESCRIPTION_LENGTH,
            DESCRIPTION_HEIGHT
        );
    }
}

/**
 * Set labels on the scale.
 *
 * @param x horizontal coordinate.
 * @param y vertical coordinate.
 * @param year the label text.

```



```

    * @param label the label to position.
    */
private void positionLabel(
    int x,
    int y,
    int year,
    JLabel label
) {
    label.setBounds(
        x + LABEL_OFFSET,
        y + LABEL_OFFSET,
        LABEL_LENGTH,
        LABEL_HEIGHT
    );
    label.setText(String.valueOf(year));
}

/**
 * Adds an EntityDisplay to the TimeLine.
 *
 * @param x horizontal coordinate.
 * @param y vertical coordinate.
 * @param ed the EntityDisplay.
 */
private void positionDisplay(
    int x,
    int y,
    EntityDisplay ed
) {
    int w = ed.getWidth();
    int h = ed.getHeight();
    ed.setBounds(
        x - (w / 2),
        y,
        w,
        h
    );
}

/**
 * Paints the scale onto the TimeLine.
 *
 * @param g the Graphics component.
 */
private void paintScale(Graphics g) {
    g.setColor(Color.BLACK);
    g.drawLine(
        LINE_INDENT,
        vertical,
        LINE_INDENT + lineLength,
        vertical
    );
    int x, y;
    y = vertical;
    for(int i = 0; i < intervals.length; i++) {
        x = LINE_INDENT + SCALE_INDENT + (int) (((double) scaleLength /
(intervals.length - 1)) * i);
        g.drawLine(
            x,
            y - UPNOTCH,
            x,

```

```

        y + DOWNNOTCH
    );
    positionLabel(
        x,
        y,
        intervals[i],
        labels[i]
    );
}

private int timeLineWidth;
private Integer firstYear;
private Integer lastYear;
private Integer thisYear;

/**
 * Paints the entities onto the TimeLine.
 *
 * @param g the Graphics component.
 */
private void paintEntities(Graphics g) {

    firstYear = intervals[0];
    lastYear = intervals[intervals.length - 1];

    int timeSpan = lastYear - firstYear;
    double ratio = ((double) scaleLength / timeSpan);

    boolean updateLocations = false;
    if (this.getSize().width != timeLineWidth) {
        updateLocations = true;
        timeLineWidth = this.getSize().width;
    }

    for (EntityDisplay eDisplay : eDisplays) {
        thisYear = eDisplay.getYear();
        if (thisYear != null) {
            int timePosition = thisYear - firstYear;
            int x, y;
            x = LINE_INDENT + SCALE_INDENT + (int) (timePosition *
ratio);
            y = vertical - IMAGE_UPPER;

            g.setColor(Color.BLACK);
            g.drawLine(x, y + eDisplay.getHeight(), x, vertical);
            g.fillOval(x - RADIUS, vertical - RADIUS, DIAMETER,
DIAMETER);

            if (updateLocations) {
                positionDisplay(x, y, eDisplay);
            }
        }
    }

}

/**
 * Return the name of this TimeLine.
 */

```

```

        * @return the name.
        */
@Override
public String getName() {
    return name;
}

/**
 * Returns the Entities displayed by this TimeLine.
 *
 * @return the Entities.
 */
public ManMadeObject[] getEntities() {
    return objects;
}

/**
 * Notifies the TimeLine that an update is necessary.
 */
public void setUpdate() {
    update = true;
}
}

```

12.1.7 TimeLineCollection.java

```
package org.jghill.timelinesvisualizercollections.display;

import java.awt.Color;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import static java.util.Comparator.comparingInt;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;
import org.jghill.timelinevisualizerentities.ManMadeObject;
import javax.swing.ComboBoxModel;
import javax.swing.DefaultComboBoxModel;
import static java.util.Map.Entry.comparingByValue;
import static java.util.stream.Collectors.toMap;

/**
 * Filters a collection of objects to a set of TimeLines.
 *
 * @author JGHill
 */
public class TimeLineCollection {

    private static final List<Color> COLORS = Colours.getColours();

    private final String none = "None";
    private final String query = "Query";
    private final String source = "Source";
    private final String depicts = "Depicts";
    private final String material = "Material";
    private final String type = "Type";
    private final String technique = "Technique";
    private final String creator = "Creator";

    private final int MAX_CATEGORIES = 4;

    private final CollectionDisplayPanel cdp;

    private TimeLine[] timeLines;

    private final TreeMap<String, TreeMap<String, Integer>> filters;
    private TreeMap<String, TreeMap<String, Integer>> filterList;

    /**
     * Constructor.
     */
    public TimeLineCollection(CollectionDisplayPanel cdp) {
        this.cdp = cdp;
        filters = new TreeMap<>();
    }

    /**
     * Selects the filters that will be used to fill the ComboBox.
     *
     * @param collection of objects.
     */
    public void createFilters(ManMadeObject[] collection) {
```

```

        createInitialFilter();
        for (ManMadeObject object: collection) {
            if (object.getTimeSpan() != null) {

                addToFilter(object.getQueryName(), query);
                addToFilter(object.getSourceName(), source);
                addToFilter(object.getDepicts(), depicts);
                addToFilter(object.getConsists(), material);
                addToFilter(object.getType(), type);
                addToFilter(object.getTechnique(), technique);
                addToFilter(object.getCreator(), creator);

            }
        }

        filterList = new TreeMap<>();
        filterList.put(none, new TreeMap<>());
        filters.forEach((k, v) -> {
            if (v.size() > 1) {
                filterList.put(k, v);
            }
        });
    }

    /**
    * Adds a dimension and adds to the count if the dimension already has
been
    * entered.
    *
    * @param objectDimension
    * @param filterName
    */
    private void addToFilter(
        String objectDimension,
        String filterName
    ) {
        if (!filters.get(filterName).containsKey(objectDimension)) {
            filters.get(filterName).put(
                objectDimension,
                1
            );
        } else {
            filters.get(filterName).put(
                objectDimension,
                filters.get(filterName).get(objectDimension) + 1
            );
        }
    }

    /**
    * Chooses the option to filter the results by.
    *
    * @return the option.
    */
    public String chooseFilter() {

        String choice = "None";
        int maxScore = 0;

```

```

        for (Map.Entry<String, TreeMap<String, Integer>> entry :
filterList.entrySet()) {
            int tempScore = 1;
            for (Map.Entry<String, Integer> dimension :
entry.getValue().entrySet()) {
                tempScore = (tempScore * dimension.getValue());
            }
            tempScore = tempScore * Math.min(
                MAX_CATEGORIES - 1,
                entry.getValue().entrySet().size()
            );
            if (entry.getKey().equalsIgnoreCase(query) ||
                entry.getKey().equalsIgnoreCase(source)) {
                int penalty = 2;
                tempScore = tempScore / penalty;
            }
            if (tempScore > maxScore) {
                choice = entry.getKey();
                maxScore = tempScore;
            }
        }

        return choice;
    }

    /**
     * Create TimeLines from a set of objects.
     *
     * @param collection of objects.
     * @return an array of TimeLines.
     */
    public void createTimeLines(
        ManMadeObject[] collection,
        String filter
    ) {
        Collections.shuffle(COLORS);
        runFilter(collection, filter);
    }

    /**
     * Filters the results.
     */
    private void runFilter(
        ManMadeObject[] collection,
        String filter
    ) {
        Map<String, List<ManMadeObject>> categories = new TreeMap<>();

        for (ManMadeObject object: collection) {
            if (object.getTimeSpan() != null) {
                String result;
                switch(filter) {
                    case query :
                        result = object.getQueryName();
                        break;
                    case source :
                        result = object.getSourceName();
                        break;
                    case depicts :

```

```

        result = object.getDepicts();
        break;
    case material :
        result = object.getConsists();
        break;
    case type :
        result = object.getType();
        break;
    case technique :
        result = object.getTechnique();
        break;
    case creator :
        result = object.getCreator();
        break;
    case none :
        result = "General";
        break;
    default :
        result = "General";
        break;
    }

    if (!result.equalsIgnoreCase("")) {
        List<ManMadeObject> list;
        if (categories.containsKey(result)) {
            list = categories.get(result);
            list.add(object);
        } else {
            list = new ArrayList<>();
            list.add(object);
            categories.putIfAbsent(result, list);
        }
    } else {
        List<ManMadeObject> list;
        if (categories.containsKey("Blank")) {
            list = categories.get("Blank");
            list.add(object);
        } else {
            list = new ArrayList<>();
            list.add(object);
            categories.putIfAbsent("Blank", list);
        }
    }

    }

    timeLines = createTimeLineArray(categories);
    categories = sortCategories(categories);
    assignTimeLines(categories);

}

/**
 * Creates a TimeLine array based on the number of categories, capping
the
 * number if it is over 4.
 *
 * @param categories
 * @return an array of TimeLines.
 */

```

```

private TimeLine[] createTimeLineArray(
    Map<String, List<ManMadeObject>> categories
) {
    TimeLine[] timeline;
    int categoriesCount = categories.size();
    if (categoriesCount <= MAX_CATEGORIES) {
        timeline = new TimeLine[categoriesCount];
    } else {
        timeline = new TimeLine[MAX_CATEGORIES];
    }
    return timeline;
}

/**
 * Sorts the TimeLines by their length.
 *
 * @param categories a list of categories.
 * @return a sorted TreeMap of categories.
 */
private Map<String, List<ManMadeObject>> sortCategories(
    Map<String, List<ManMadeObject>> categories
) {
    Map<String, List<ManMadeObject>> sorted;
    sorted = categories.entrySet().stream()
        .sorted(comparingByValue(comparingInt((List list) ->
list.size()).reversed()))
        .collect(toMap(
            Map.Entry::getKey,
            Map.Entry::getValue,
            (a, b) -> {throw new AssertionError();},
            LinkedHashMap::new
        ));
    return sorted;
}

/**
 * Creates all TimeLines from the passed categories.
 *
 * @param categories that can be passed into TimeLines.
 */
private void assignTimeLines(
    Map<String, List<ManMadeObject>> categories
) {
    int count = 0;
    ArrayList<ManMadeObject> other = new ArrayList<>();

    for (Map.Entry<String, List<ManMadeObject>> entry:
categories.entrySet()) {
        if (count < MAX_CATEGORIES - 1) {
            timeLines[count] = new TimeLine(
                entry.getKey(),
                entry.getValue().toArray(new
ManMadeObject[entry.getValue().size()]),
                COLORS.get(count),
                cdp
            );
            count++;
        } else {
            other.addAll(entry.getValue());
        }
    }
}

```



```

        if (count >= (MAX_CATEGORIES - 1) && !other.isEmpty()) {
            timeLines[MAX_CATEGORIES - 1] = new TimeLine(
                "Other",
                other.toArray(new ManMadeObject[0]),
                COLORS.get(count),
                cdp
            );
        }
    }

    /**
     * Returns the man made objects associated with the given name.
     *
     * @param timelineName the name of the TimeLine to return objects from.
     * @return the objects associated with the TimeLine.
     */
    public ManMadeObject[] getTimeLineObjects(String timelineName) {
        ManMadeObject[] objects = null;
        if (timelineName.equalsIgnoreCase("None")) {
            objects = getAllTimeLineObjects();
        } else {
            for (TimeLine timeline : timeLines) {
                if (timeline.getName().equalsIgnoreCase(timelineName)) {
                    objects = timeline.getEntities();
                }
            }
        }
        return objects;
    }

    /**
     * Returns the man made objects associated with the given name.
     *
     * @param timelineName the name of the TimeLine to return objects from.
     * @return the objects associated with the TimeLine.
     */
    public ManMadeObject[] getAllTimeLineObjects() {
        List<ManMadeObject> objectList = new ArrayList<>();
        for (TimeLine timeline : timeLines) {
            objectList.addAll(Arrays.asList(timeline.getEntities()));
        }
        return objectList.toArray(new ManMadeObject[0]);
    }

    /**
     * Return an array of the TimeLines.
     *
     * @return an array of TimeLines.
     */
    public TimeLine[] getTimeLines() {
        return timeLines;
    }

    /**
     * Returns an array of names of the TimeLines.
     *
     * @return the names of the TimeLines.
     */
    private String[] getTimeLinesNames() {

```

```

        String[] timeLineNames;
        timeLineNames = new String[timeLines.length + 1];
        timeLineNames[0] = "None";
        for (int i = 1; i < timeLineNames.length; i++) {
            timeLineNames[i] = timeLines[i-1].getName();
        }
        return timeLineNames;
    }

    /**
     * Get a count of the timeLines in this collection.
     *
     * @return a count of TimeLines.
     */
    public int getCount() {
        return timeLines.length;
    }

    /**
     * Returns the comboBox model for the filters.
     *
     * @return the comboBox model.
     */
    public ComboBoxModel getFilterComboBoxModel() {
        return new DefaultComboBoxModel(filterList.keySet().toArray());
    }

    /**
     * Returns the comboBox model for the parameters.
     *
     * @return the comboBox model.
     */
    public ComboBoxModel getCategoryComboBoxModel() {
        return new DefaultComboBoxModel(getTimeLinesNames());
    }

    /**
     * Clears the filter.
     */
    private void createInitialFilter() {
        filters.clear();
        filters.put(none, new TreeMap<>());
        filters.put(query, new TreeMap<>());
        filters.put(source, new TreeMap<>());
        filters.put(depicts, new TreeMap<>());
        filters.put(material, new TreeMap<>());
        filters.put(type, new TreeMap<>());
        filters.put(technique, new TreeMap<>());
        filters.put(creator, new TreeMap<>());
    }
}

```

12.1.8 CollectionTopComponent.java

```
package org.jghill.timelinesvisualizercollections.gui;

import java.awt.Cursor;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.awt.event.KeyEvent;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;
import java.util.regex.Pattern;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JComboBox;
import javax.xml.parsers.ParserConfigurationException;
import org.apache.jena.atlas.web.HttpException;
import org.netbeans.api.settings.ConvertAsProperties;
import org.openide.awt.ActionID;
import org.openide.windows.TopComponent;
import org.openide.util.NbBundle.Messages;
import org.jghill.timelinesvisualizercollections.Collection;
import
org.jghill.timelinesvisualizercollections.container.CollectionContainer;
import org.jghill.timelinesvisualizercollections.display.EntityDisplay;
import org.jghill.timelinesvisualizercollections.display.TimeLineCollection;
import org.jghill.timelinesvisualizercollectionxml.CollectionXMLWriter;
import org.jghill.timelinesvisualizercollectionxml.CollectionXMLWriterImpl;
import org.jghill.timelinesvisualizerdispatcher.Dispatcher;
import org.jghill.timelinesvisualizerqueriesbuilder.QueryBuilder;
import org.jghill.timelinesvisualizerqueriesbuilder.QuerySettings;
import org.jghill.timelinevisualizerentities.Entities;
import org.jghill.timelinevisualizerentities.ManMadeObject;
import org.jghill.timelinevisualizerentitiescollection.EntitiesCollection;
import org.jghill.timelinevisualizerqueries.QueryShell;
import org.jghill.timelinevisualizersources.Source;
import org.jghill.timelinevisualizersources.SourceCollection;
import org.netbeans.api.io.IOException;
import org.netbeans.api.io.InputOutput;
import org.openide.DialogDisplayer;
import org.openide.NotifyDescriptor;
import org.openide.util.Exceptions;
import org.openide.util.Lookup;
import org.openide.util.LookupEvent;
import org.openide.util.RequestProcessor;
import org.openide.util.lookup.AbstractLookup;
import org.openide.util.lookup.InstanceContent;

/**
 * A window for displaying a collection and it's internals.
 */
@ConvertAsProperties(
    dtd = "-//org.jghill.timelinesvisualizercollectionsgui//Collection//EN",
    autostore = false
)
@TopComponent.Description(
    preferredID = "CollectionTopComponent"
)
@TopComponent.Registration(mode = "editor", openAtStartup = false)
@ActionID(category = "Window", id =
"org.jghill.timelinesvisualizercollectionsgui.CollectionTopComponent")
```

```

@TopComponent.OpenActionRegistration(
    displayName = "#CTL_CollectionAction",
    preferredID = "CollectionTopComponent"
)
@Messages({
    "CTL_CollectionAction=Collection",
    "CTL_CollectionTopComponent=Collection Window",
    "HINT_CollectionTopComponent=This is a Collection window"
})
public final class CollectionTopComponent extends TopComponent implements
FocusListener {

    private static final int TAB_VISUAL = 3;

    /**
     * Constructor.
     */
    public CollectionTopComponent() {

        Lookup tcLookup = CollectionContainer.getLookup();
        coll = tcLookup.lookup(Collection.class);
        CollectionContainer.addCollection(coll);

        initComponents();
        setToolTipText(Bundle.HINT_CollectionTopComponent());
        putClientProperty(TopComponent.PROP_CLOSING_DISABLED,
Boolean.FALSE);
        putClientProperty(TopComponent.PROP_DRAGGING_DISABLED,
Boolean.TRUE);
        putClientProperty(TopComponent.PROP_MAXIMIZATION_DISABLED,
Boolean.TRUE);
        putClientProperty(TopComponent.PROP_UNDOCKING_DISABLED,
Boolean.TRUE);

        TitleTextBox.setText(coll.getName());
        qtb = (QueryTableModel) QueriesTable.getModel();
        etb = (EntityTableModel) EntitiesTable.getModel();

        Lookup sLookup = SourceCollection.getInstance().getLookup();
        sources = sLookup.lookupResult(Source.class);
        sources.allInstances();
        sources.addLookupListener((LookupEvent e) -> {
            SourceComboBox.setModel(
                new DefaultComboBoxModel(
                    SourceCollection.collectionToArray()));
        });

        collectionDisplayPanel.setSlider(ZoomSlider);

        timeLinesCollection = new TimeLineCollection(
            collectionDisplayPanel
        );

    }

    /**
     * This method is called from within the constructor to initialise the
    form.
     * WARNING: Do NOT modify this code. The content of this method is
    always
     * regenerated by the Form Editor.

```

```

*/
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    Tab = new javax.swing.JTabbedPane();
    Information = new javax.swing.JPanel();
    NotesTextPanel = new javax.swing.JScrollPane();
    NotesTextBox = new javax.swing.JTextArea();
    NotesText = new javax.swing.JLabel();
    TitleTextBox = new javax.swing.JTextField();
    Queries = new javax.swing.JPanel();
    QueriesScrollPane = new javax.swing.JScrollPane();
    QueriesTable = new javax.swing.JTable();
    ExistingQueriesText = new javax.swing.JLabel();
    QuerySeparator = new javax.swing.JSeparator();
    QueryBuilderText = new javax.swing.JLabel();
    CreateButton = new javax.swing.JButton();
    ResetButton = new javax.swing.JButton();
    VerticalSeparator = new javax.swing.JSeparator();
    SourceTextLabel = new javax.swing.JLabel();
    SourceComboBox = new javax.swing.JComboBox<>();
    HasNameCheckBox = new javax.swing.JCheckBox();
    HasImageCheckBox = new javax.swing.JCheckBox();
    HasIdentifierCheckBox = new javax.swing.JCheckBox();
    DeleteButton = new javax.swing.JButton();
    AndText1 = new javax.swing.JLabel();
    NameTextField = new javax.swing.JTextField();
    IdentifierTextField = new javax.swing.JTextField();
    RunButton = new javax.swing.JButton();
    HasLimitCheckBox = new javax.swing.JCheckBox();
    LimitTextField = new javax.swing.JTextField();
    SourceNameTextLabel = new javax.swing.JLabel();
    QueryNameTextField = new javax.swing.JTextField();
    jSeparator1 = new javax.swing.JSeparator();
    CreationYearLabel = new javax.swing.JLabel();
    CreationStartYearTextField = new javax.swing.JTextField();
    CreationEndYearTextField = new javax.swing.JTextField();
    HasDepictionCheckBox = new javax.swing.JCheckBox();
    DepictionTextField = new javax.swing.JTextField();
    HasConsistsCheckBox = new javax.swing.JCheckBox();
    MaterialTextField = new javax.swing.JTextField();
    HasTypeCheckBox = new javax.swing.JCheckBox();
    TypeTextField = new javax.swing.JTextField();
    HasTechniqueCheckBox = new javax.swing.JCheckBox();
    TechniqueTextField = new javax.swing.JTextField();
    CreatedByCheckBox = new javax.swing.JCheckBox();
    CreatedByTextField = new javax.swing.JTextField();
    Entities = new javax.swing.JPanel();
    EntitiesScrollPane = new javax.swing.JScrollPane();
    EntitiesTable = new javax.swing.JTable();
    Visualizer = new javax.swing.JPanel();
    FirstFilterLabel = new javax.swing.JLabel();
    FirstFilterComboBox = new javax.swing.JComboBox<>();
    Group1Label = new javax.swing.JLabel();
    Group1ComboBox = new javax.swing.JComboBox<>();
    Group1FilterLabel = new javax.swing.JLabel();
    Group1FilterComboBox = new javax.swing.JComboBox<>();
    Group2Label = new javax.swing.JLabel();
    Group2ComboBox = new javax.swing.JComboBox<>();
    Group2FilterLabel = new javax.swing.JLabel();
    Group2FilterComboBox = new javax.swing.JComboBox<>();

```



```

    );
    InformationLayout.setVerticalGroup(

InformationLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(InformationLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(TitleTextBox,
javax.swing.GroupLayout.PREFERRED_SIZE, 50,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(NotesText)
        .addGap(18, 18, 18)
        .addComponent(NotesTextPanel,
javax.swing.GroupLayout.DEFAULT_SIZE, 352, Short.MAX_VALUE)
        .addContainerGap()
    );

Tab.addTab(org.openide.util.NbBundle.getMessage(CollectionTopComponent.class, "CollectionTopComponent.Information.TabConstraints.tabTitle"),
Information); // NOI18N

    QueriesScrollPane.setPreferredSize(new java.awt.Dimension(440,
400));

    QueriesTable.setAutoCreateRowSorter(true);
    QueriesTable.setModel(new QueryTableModel(coll));
    QueriesTable.setRowHeight(20);
    QueriesScrollPane.setViewportView(QueriesTable);

    org.openide.awt.Mnemonics.setLocalizedText(ExistingQueriesText,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.ExistingQueriesText.text")); // NOI18N
    ExistingQueriesText.setPreferredSize(new java.awt.Dimension(100,
14));

    QuerySeparator.setOrientation(javax.swing.SwingConstants.VERTICAL);

    org.openide.awt.Mnemonics.setLocalizedText(QueryBuilderText,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.QueryBuilderText.text")); // NOI18N

    org.openide.awt.Mnemonics.setLocalizedText(CreateButton,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.CreateButton.text")); // NOI18N
    CreateButton.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        CreateButtonActionPerformed(evt);
    }
});

    org.openide.awt.Mnemonics.setLocalizedText(ResetButton,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.ResetButton.text")); // NOI18N
    ResetButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        ResetButtonActionPerformed(evt);
    }
});

```

```

        org.openide.awt.Mnemonics.setLocalizedText(SourceTextLabel,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.SourceTextLabel.text")); // NOI18N

        SourceComboBox.setModel(SourceCollection.getSourceComboBoxModel());

        org.openide.awt.Mnemonics.setLocalizedText(HasNameCheckBox,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.HasNameCheckBox.text")); // NOI18N
        HasNameCheckBox.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                HasNameCheckBoxActionPerformed(evt);
            }
        });

        HasImageCheckBox.setSelected(true);
        org.openide.awt.Mnemonics.setLocalizedText(HasImageCheckBox,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.HasImageCheckBox.text")); // NOI18N

        org.openide.awt.Mnemonics.setLocalizedText(HasIdentifierCheckBox,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.HasIdentifierCheckBox.text")); // NOI18N
        HasIdentifierCheckBox.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                HasIdentifierCheckBoxActionPerformed(evt);
            }
        });

        org.openide.awt.Mnemonics.setLocalizedText(DeleteButton,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.DeleteButton.text")); // NOI18N
        DeleteButton.addActionListener(new java.awt.event.ActionListener()
{
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                DeleteButtonActionPerformed(evt);
            }
        });

        org.openide.awt.Mnemonics.setLocalizedText(AndText1,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.AndText1.text")); // NOI18N

NameTextField.setText(org.openide.util.NbBundle.getMessage(CollectionTopCom
ponent.class, "CollectionTopComponent.NameTextField.text")); // NOI18N
        NameTextField.setEnabled(false);

IdentifierTextField.setText(org.openide.util.NbBundle.getMessage(Collection
TopComponent.class, "CollectionTopComponent.IdentifierTextField.text")); //
NOI18N
        IdentifierTextField.setEnabled(false);

        org.openide.awt.Mnemonics.setLocalizedText(RunButton,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.RunButton.text")); // NOI18N
        RunButton.addActionListener(new java.awt.event.ActionListener() {

```



```

        public void actionPerformed(java.awt.event.ActionEvent evt) {
            RunButtonActionPerformed(evt);
        }
    });

    HasLimitCheckBox.setSelected(true);
    org.openide.awt.Mnemonics.setLocalizedText (HasLimitCheckBox,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.HasLimitCheckBox.text")); // NOI18N
    HasLimitCheckBox.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            HasLimitCheckBoxActionPerformed(evt);
        }
    });

    LimitTextField.setHorizontalAlignment (javax.swing.JTextField.RIGHT);

    LimitTextField.setText (org.openide.util.NbBundle.getMessage(CollectionTopCo
mponent.class, "CollectionTopComponent.LimitTextField.text")); // NOI18N

    LimitTextField.setToolTipText (org.openide.util.NbBundle.getMessage(Collecti
onTopComponent.class, "CollectionTopComponent.LimitTextField.toolTipText"));
// NOI18N

    org.openide.awt.Mnemonics.setLocalizedText (SourceNameTextLabel,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.SourceNameTextLabel.text")); // NOI18N
    SourceNameTextLabel.setOpaque(true);

    QueryNameTextField.setText (org.openide.util.NbBundle.getMessage(CollectionT
opComponent.class, "CollectionTopComponent.QueryNameTextField.text")); //
NOI18N
    QueryNameTextField.setNextFocusableComponent (SourceComboBox);

    org.openide.awt.Mnemonics.setLocalizedText (CreationYearLabel,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.CreationYearLabel.text")); // NOI18N

    CreationYearLabel.setToolTipText (org.openide.util.NbBundle.getMessage(Colle
ctionTopComponent.class,
"CollectionTopComponent.CreationYearLabel.toolTipText")); // NOI18N

    CreationStartYearTextField.setText (org.openide.util.NbBundle.getMessage(Col
lectionTopComponent.class,
"CollectionTopComponent.CreationStartYearTextField.text")); // NOI18N

    CreationEndYearTextField.setText (org.openide.util.NbBundle.getMessage(Colle
ctionTopComponent.class,
"CollectionTopComponent.CreationEndYearTextField.text")); // NOI18N

    org.openide.awt.Mnemonics.setLocalizedText (HasDepictionCheckBox,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.HasDepictionCheckBox.text")); // NOI18N
    HasDepictionCheckBox.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            HasDepictionCheckBoxActionPerformed(evt);
        }
    });

```

```

        }
    });

    DepictionTextField.setText(org.openide.util.NbBundle.getMessage(CollectionTopComponent.class, "CollectionTopComponent.DepictionTextField.text")); // NOI18N
    DepictionTextField.setEnabled(false);

    org.openide.awt.Mnemonics.setLocalizedText(HasConsistsCheckBox,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.HasConsistsCheckBox.text")); // NOI18N
    HasConsistsCheckBox.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            HasConsistsCheckBoxActionPerformed(evt);
        }
    });

    MaterialTextField.setText(org.openide.util.NbBundle.getMessage(CollectionTopComponent.class, "CollectionTopComponent.MaterialTextField.text")); // NOI18N
    MaterialTextField.setEnabled(false);

    org.openide.awt.Mnemonics.setLocalizedText(HasTypeCheckBox,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.HasTypeCheckBox.text")); // NOI18N
    HasTypeCheckBox.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            HasTypeCheckBoxActionPerformed(evt);
        }
    });

    TypeTextField.setText(org.openide.util.NbBundle.getMessage(CollectionTopComponent.class, "CollectionTopComponent.TypeTextField.text")); // NOI18N
    TypeTextField.setEnabled(false);

    org.openide.awt.Mnemonics.setLocalizedText(HasTechniqueCheckBox,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.HasTechniqueCheckBox.text")); // NOI18N
    HasTechniqueCheckBox.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            HasTechniqueCheckBoxActionPerformed(evt);
        }
    });

    TechniqueTextField.setText(org.openide.util.NbBundle.getMessage(CollectionTopComponent.class, "CollectionTopComponent.TechniqueTextField.text")); // NOI18N
    TechniqueTextField.setEnabled(false);

    org.openide.awt.Mnemonics.setLocalizedText(CreatedByCheckBox,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.CreatedByCheckBox.text")); // NOI18N
    CreatedByCheckBox.addActionListener(new
java.awt.event.ActionListener() {

```

```

        public void actionPerformed(java.awt.event.ActionEvent evt) {
            CreatedByCheckBoxActionPerformed(evt);
        }
    });

    CreatedByTextField.setText(org.openide.util.NbBundle.getMessage(CollectionTopComponent.class, "CollectionTopComponent.CreatedByTextField.text")); // NOI18N
    CreatedByTextField.setEnabled(false);

    javax.swing.GroupLayout QueriesLayout = new
    javax.swing.GroupLayout(Queries);
    Queries.setLayout(QueriesLayout);
    QueriesLayout.setHorizontalGroup(

    QueriesLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(QueriesLayout.createSequentialGroup()
            .addGroup(QueriesLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(QueriesLayout.createSequentialGroup()
                    .addGap(10, 10, 10)
                    .addComponent(ExistingQueriesText,
                        javax.swing.GroupLayout.PREFERRED_SIZE,
                        javax.swing.GroupLayout.DEFAULT_SIZE,
                        javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(QueriesLayout.createSequentialGroup()
                        .addComponent(DeleteButton)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(RunButton))
                    .addComponent(QueriesScrollPane,
                        javax.swing.GroupLayout.PREFERRED_SIZE, 300,
                        javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGroup(QueriesLayout.createSequentialGroup()
                    .addGap(10, 10, 10)
                    .addComponent(QuerySeparator,
                        javax.swing.GroupLayout.PREFERRED_SIZE,
                        javax.swing.GroupLayout.DEFAULT_SIZE,
                        javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addGroup(QueriesLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(VerticalSeparator)
                        .addGroup(QueriesLayout.createSequentialGroup()
                            .addComponent(QueryBuilderText)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(SourceNameTextLabel,
                                javax.swing.GroupLayout.PREFERRED_SIZE, 66,
                                javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addGap(18, 18, 18)
                            .addComponent(QueryNameTextField,
                                javax.swing.GroupLayout.PREFERRED_SIZE, 186,
                                javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jSeparator1,
                                javax.swing.GroupLayout.Alignment.TRAILING)
                            .addGroup(QueriesLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                                .addGroup(QueriesLayout.createSequentialGroup()
                                    .addGap(10, 10, 10)
                                    .addComponent(ResetButton)

```

```

        .addGap(18, 18, 18)
        .addComponent(CreateButton))
    .addGroup(QueriesLayout.createSequentialGroup())
    .addComponent(SourceTextLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 66,
javax.swing.GroupLayout.PREFERRED_SIZE)
    .addGap(18, 18, 18)
    .addComponent(SourceComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE, 203,
javax.swing.GroupLayout.PREFERRED_SIZE))
    .addGroup(QueriesLayout.createSequentialGroup())
    .addGroup(QueriesLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.LEADING)
        .addComponent(HasLimitCheckBox)
        .addComponent(LimitTextField,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE, 112,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGroup(QueriesLayout.createSequentialGroup())
        .addGroup(QueriesLayout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(CreationYearLabel)
            .addComponent(CreatedByCheckBox))
            .addGap(24, 24, 24)
            .addGroup(QueriesLayout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING, false)
                .addComponent(CreationStartYearTextFiel
d)
                .addComponent(CreatedByTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 112,
javax.swing.GroupLayout.PREFERRED_SIZE)))
        .addGroup(QueriesLayout.createSequentialGroup())
        .addGroup(QueriesLayout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(HasNameCheckBox)
            .addComponent(HasDepictionCheckBox)
            .addComponent(HasTypeCheckBox))
            .addGap(50, 50, 50)
            .addGroup(QueriesLayout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING, false)
                .addComponent(TypeTextField)
                .addComponent(NameTextField)
                .addComponent(DepictionTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 112,
javax.swing.GroupLayout.PREFERRED_SIZE)))
        .addGap(18, 18, 18)
        .addGroup(QueriesLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRA
ILING, QueriesLayout.createSequentialGroup())
            .addGroup(QueriesLayout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(HasConsistsCheckBox)
                .addComponent(HasImageCheckBox)
                .addGroup(QueriesLayout.createSequentia
lGroup())
                    .addComponent(AndText1)
                    .addGap(18, 18, 18)
                    .addComponent(CreationEndYearTextFi
eld, javax.swing.GroupLayout.PREFERRED_SIZE, 110,
javax.swing.GroupLayout.PREFERRED_SIZE))
            )
        )
    )
)

```

```

        .addComponent(HasIdentifierCheckBox))
        .addGap(22, 22, 22))
        .addGroup(QueriesLayout.createSequentialGroup())
        .addComponent(HasTechniqueCheckBox)
        .addGap(42, 42, 42)))
        .addGroup(QueriesLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.TRAILING, false)
        .addComponent(TechniqueTextField)
        .addComponent(MaterialTextField,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 112, Short.MAX_VALUE)
        .addComponent(IdentifierTextField,
javax.swing.GroupLayout.Alignment.LEADING)))
        .addGap(114, 114, 114))
    );
    QueriesLayout.setVerticalGroup(

    QueriesLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(QueriesLayout.createSequentialGroup())
        .addContainerGap()
        .addGroup(QueriesLayout.createParallelGroup(javax.swing.Gro
upLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
QueriesLayout.createSequentialGroup())
        .addComponent(ExistingQueriesText,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentP
lacement.RELATED)
        .addComponent(QueriesScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, 403, Short.MAX_VALUE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentP
lacement.RELATED)
        .addGroup(QueriesLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.LEADING)
        .addComponent(RunButton)
        .addGroup(QueriesLayout.createParallelGroup(jav
ax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(DeleteButton))))
        .addGroup(QueriesLayout.createSequentialGroup())
        .addGroup(QueriesLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
        .addComponent(QueryBuilderText)
        .addComponent(SourceNameTextLabel)
        .addComponent(QueryNameTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentP
lacement.RELATED)
        .addComponent(VerticalSeparator,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentP
lacement.RELATED)
        .addGroup(QueriesLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
        .addComponent(SourceComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,

```

```

javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(SourceTextLabel))
        .addGap(26, 26, 26)
        .addGroup(QueriesLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
        .addComponent(CreationEndYearTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(AndText1)
        .addComponent(CreationStartYearTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(CreationYearLabel))
        .addGap(18, 18, 18)
        .addGroup(QueriesLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
        .addComponent(NameTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(HasNameCheckBox)
        .addComponent(IdentifierTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(HasIdentifierCheckBox))
        .addGap(18, 18, 18)
        .addGroup(QueriesLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
        .addComponent(DepictionTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(HasDepictionCheckBox)
        .addComponent(MaterialTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(HasConsistsCheckBox))
        .addGap(18, 18, 18)
        .addGroup(QueriesLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
        .addComponent(HasTypeCheckBox)
        .addComponent(TypeTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(HasTechniqueCheckBox)
        .addComponent(TechniqueTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(18, 18, 18)
        .addGroup(QueriesLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
        .addComponent(CreatedByCheckBox)
        .addComponent(CreatedByTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,

```

```

javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent (HasImageCheckBox))
        .addGap (18, 18, 18)
        .addGroup (QueriesLayout.createParallelGroup (javax.s
wing.GroupLayout.Alignment.BASELINE)
        .addComponent (HasLimitCheckBox)
        .addComponent (LimitTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap (javax.swing.LayoutStyle.ComponentP
lacement.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        .addComponent (jSeparator1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap (javax.swing.LayoutStyle.ComponentP
lacement.RELATED)
        .addGroup (QueriesLayout.createParallelGroup (javax.s
wing.GroupLayout.Alignment.BASELINE)
        .addComponent (CreateButton)
        .addComponent (ResetButton)))
        .addContainerGap ()
        .addComponent (QuerySeparator,
javax.swing.GroupLayout.Alignment.TRAILING)
    );

```

```

Tab.addTab (org.openide.util.NbBundle.getMessage (CollectionTopComponent.clas
s, "CollectionTopComponent.Queries.TabConstraints.tabTitle"), Queries); //
NOI18N

```

```

EntitiesScrollPane.setHorizontalScrollBarPolicy (javax.swing.ScrollPaneConst
ants.HORIZONTAL_SCROLLBAR_NEVER);

```

```

    EntitiesTable.setAutoCreateRowSorter (true);
    EntitiesTable.setModel (new EntityTableModel (coll));

```

```

EntitiesTable.setAutoResizeMode (javax.swing.JTable.AUTO_RESIZE_NEXT_COLUMN);
EntitiesScrollPane.setViewportView (EntitiesTable);

```

```

    javax.swing.GroupLayout EntitiesLayout = new
javax.swing.GroupLayout (Entities);
    Entities.setLayout (EntitiesLayout);
    EntitiesLayout.setHorizontalGroup (

```

```

EntitiesLayout.createParallelGroup (javax.swing.GroupLayout.Alignment.LEADING)

```

```

        .addGroup (EntitiesLayout.createSequentialGroup ()
        .addContainerGap ()
        .addComponent (EntitiesScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, 925, Short.MAX_VALUE)
        .addContainerGap ())
    );

```

```

    EntitiesLayout.setVerticalGroup (

```

```

EntitiesLayout.createParallelGroup (javax.swing.GroupLayout.Alignment.LEADING)

```

```

        .addGroup (EntitiesLayout.createSequentialGroup ()

```

```

        .addContainerGap()
        .addComponent(EntitiesScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, 452, Short.MAX_VALUE)
        .addContainerGap()
    );

Tab.addTab(org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.Entities.TabConstraints.tabTitle"), Entities);
// NOI18N

    org.openide.awt.Mnemonics.setLocalizedText(FirstFilterLabel,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.FirstFilterLabel.text")); // NOI18N
    FirstFilterLabel.setOpaque(true);

FirstFilterComboBox.setToolTipText(org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.FirstFilterComboBox.toolTipText")); // NOI18N
    FirstFilterComboBox.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            FirstFilterComboBoxActionPerformed(evt);
        }
    });

    org.openide.awt.Mnemonics.setLocalizedText(Group1Label,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.Group1Label.text")); // NOI18N
    Group1Label.setOpaque(true);

    Group1ComboBox.setEnabled(false);
    Group1ComboBox.addActionListener(new java.awt.event.ActionListener()
{
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            Group1ComboBoxActionPerformed(evt);
        }
    });

    org.openide.awt.Mnemonics.setLocalizedText(Group1FilterLabel,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.Group1FilterLabel.text")); // NOI18N
    Group1FilterLabel.setOpaque(true);

    Group1FilterComboBox.setEnabled(false);
    Group1FilterComboBox.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            Group1FilterComboBoxActionPerformed(evt);
        }
    });

    org.openide.awt.Mnemonics.setLocalizedText(Group2Label,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.Group2Label.text")); // NOI18N
    Group2Label.setOpaque(true);

    Group2ComboBox.setEnabled(false);
    Group2ComboBox.addActionListener(new java.awt.event.ActionListener()
{

```



```

        public void actionPerformed(java.awt.event.ActionEvent evt) {
            Group2ComboBoxActionPerformed(evt);
        }
    });

    org.openide.awt.Mnemonics.setLocalizedText(Group2FilterLabel,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.Group2FilterLabel.text")); // NOI18N
    Group2FilterLabel.setOpaque(true);

    Group2FilterComboBox.setEnabled(false);
    Group2FilterComboBox.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            Group2FilterComboBoxActionPerformed(evt);
        }
    });

CollectionDisplayScrollPane.setHorizontalScrollBarPolicy(javax.swing.Scroll
PaneConstants.HORIZONTAL_SCROLLBAR_ALWAYS);
    CollectionDisplayScrollPane.setViewportViewView(collectionDisplayPanel);

    ZoomSlider.setMajorTickSpacing(1);
    ZoomSlider.setMaximum(10);
    ZoomSlider.setPaintTicks(true);
    ZoomSlider.setSnapToTicks(true);
    ZoomSlider.setValue(0);

    org.openide.awt.Mnemonics.setLocalizedText(ZoomOutLabel,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.ZoomOutLabel.text")); // NOI18N

    org.openide.awt.Mnemonics.setLocalizedText(ZoomInLabel,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.ZoomInLabel.text")); // NOI18N

    org.openide.awt.Mnemonics.setLocalizedText(SaveButton,
org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.SaveButton.text")); // NOI18N
    SaveButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            SaveButtonActionPerformed(evt);
        }
    });

    javax.swing.GroupLayout VisualizerLayout = new
javax.swing.GroupLayout(Visualizer);
    Visualizer.setLayout(VisualizerLayout);
    VisualizerLayout.setHorizontalGroup(

VisualizerLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(VisualizerLayout.createSequentialGroup()
            .addGroup(VisualizerLayout.createParallelGroup(javax.swing.
GroupLayout.Alignment.LEADING)
                .addGroup(VisualizerLayout.createSequentialGroup()
                    .addContainerGap()
                    .addComponent(CollectionDisplayScrollPane))
                .addGroup(VisualizerLayout.createSequentialGroup()
                    .addGap(20, 20, 20)

```

```

        .addGroup(VisualizerLayout.createParallelGroup(java
x.swing.GroupLayout.Alignment.LEADING)
        .addGroup(VisualizerLayout.createSequentialGrou
p()
        .addComponent(FirstFilterLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 75,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.Co
mponentPlacement.RELATED)
        .addComponent(FirstFilterComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(32, 32, 32)
        .addComponent(ZoomOutLabel)
        .addGap(18, 18, 18)
        .addComponent(ZoomSlider,
javax.swing.GroupLayout.PREFERRED_SIZE, 494,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(ZoomInLabel)
        .addGap(0, 0, Short.MAX_VALUE))
        .addGroup(VisualizerLayout.createSequentialGrou
p()
        .addComponent(Group1Label,
javax.swing.GroupLayout.PREFERRED_SIZE, 75,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.Co
mponentPlacement.RELATED)
        .addComponent(Group1ComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(Group1FilterLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 75,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.Co
mponentPlacement.RELATED)
        .addComponent(Group1FilterComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(Group2Label,
javax.swing.GroupLayout.PREFERRED_SIZE, 75,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.Co
mponentPlacement.RELATED)
        .addComponent(Group2ComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(Group2FilterLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 75,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.Co
mponentPlacement.RELATED)
        .addComponent(Group2FilterComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.Co
mponentPlacement.RELATED, 88, Short.MAX_VALUE)
        .addComponent(SaveButton))))

```

```

        .addContainerGap())
    );
    VisualizerLayout.setVerticalGroup(
VisualizerLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(VisualizerLayout.createSequentialGroup()
            .addContainerGap()
            .addComponent(CollectionDisplayScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, 384, Short.MAX_VALUE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
.RELATED)
            .addGroup(VisualizerLayout.createParallelGroup(javax.swing.
GroupLayout.Alignment.LEADING)
                .addGroup(VisualizerLayout.createParallelGroup(javax.sw
ing.GroupLayout.Alignment.TRAILING)
                    .addComponent(ZoomSlider,
javax.swing.GroupLayout.PREFERRED_SIZE, 21,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(VisualizerLayout.createParallelGroup(java
x.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(FirstFilterLabel)
                        .addComponent(FirstFilterComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addComponent(ZoomInLabel,
javax.swing.GroupLayout.Alignment.LEADING))
                        .addComponent(ZoomOutLabel))
                .addGap(18, 18, 18)
                .addGroup(VisualizerLayout.createParallelGroup(javax.swing.
GroupLayout.Alignment.LEADING)
                    .addGroup(VisualizerLayout.createParallelGroup(javax.sw
ing.GroupLayout.Alignment.BASELINE)
                        .addComponent(Group2Label)
                        .addComponent(Group2ComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(Group2FilterLabel)
                        .addComponent(Group2FilterComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(SaveButton))
                    .addGroup(VisualizerLayout.createParallelGroup(javax.sw
ing.GroupLayout.Alignment.BASELINE)
                        .addComponent(Group1Label)
                        .addComponent(Group1ComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(Group1FilterLabel)
                        .addComponent(Group1FilterComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addGap(18, 18, 18))
    );

```

```

Tab.addTab(org.openide.util.NbBundle.getMessage(CollectionTopComponent.class, "CollectionTopComponent.Visualizer.TabConstraints.tabTitle"),
Visualizer); // NOI18N

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(

            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGap(10, 10, 10)
                    .addComponent(Tab, javax.swing.GroupLayout.PREFERRED_SIZE,
950, Short.MAX_VALUE)
                    .addGap(10, 10, 10)
                )
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(Tab)
                )
        );

        layout.setVerticalGroup(

            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGap(10, 10, 10)
                    .addComponent(Tab)
                    .addGap(10, 10, 10)
                )
        );

        getAccessibleContext().setAccessibleName(org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.AccessibleContext.accessibleName")); // NOI18N

        getAccessibleContext().setAccessibleDescription(org.openide.util.NbBundle.getMessage(CollectionTopComponent.class,
"CollectionTopComponent.AccessibleContext.accessibleDescription")); //
NOI18N
    } // </editor-fold>

    private void ResetButtonActionPerformed(java.awt.event.ActionEvent evt)
    {
        resetQueryBuilder();
    }

    private void CreateButtonActionPerformed(java.awt.event.ActionEvent evt)
    {
        if(querySettingsAreValid()) {
            createQuery();
        } else {
            String msg = "The query has been setup incorrectly.";
            NotifyDescriptor nd;
            nd = new NotifyDescriptor.Message(
                msg,
                NotifyDescriptor.INFORMATION_MESSAGE
            );
            DialogDisplayer.getDefault().notify(nd);
        }
    }

    private void RunButtonActionPerformed(java.awt.event.ActionEvent evt) {
        int queriesCount = coll.getQueriesCollection().getCount();
        if (queriesCount > 0) {

            String wait = "The query may take some time. " +
                "You will be notified when it is complete. " +

```

```

        "Do you wish to continue now?.";
        NotifyDescriptor waitNotification;
        waitNotification = new NotifyDescriptor.Confirmation(
            wait,
            "Continue?",
            NotifyDescriptor.OK_CANCEL_OPTION
        );
        if (DialogDisplayer.getDefault().notify(waitNotification) ==
NotifyDescriptor.OK_OPTION) {

            RequestProcessor executor = new
RequestProcessor(coll.getName());
            try
            {

this.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
                etb.clearAll();
                EntitiesCollection entities =
coll.getEntitiesCollection();

                Future<EntitiesCollection> result;

                Dispatcher dispatcher;
                dispatcher = new
Dispatcher(coll.getQueriesCollection());
                result = executor.submit(dispatcher);

                if (result != null) {

                    EntitiesCollection newEntities;
                    try
                    {
                        newEntities = result.get();
                        entities.addThing(newEntities);
                    }
                    catch(InterruptedException | ExecutionException e){}

                    entityModelChange();

                    setFirstFilter();
                    runFirstFilter();

                    NotifyDescriptor nd;
                    if (etb.getFlattenedCollection().length > 0) {
                        String msg = "Query has completed.";
                        nd = new NotifyDescriptor.Message(
                            msg,
                            NotifyDescriptor.INFORMATION_MESSAGE
                        );
                        DialogDisplayer.getDefault().notify(nd);
                        paintVisualDisplay();
                    } else {
                        String msg = "There were no objects returned.";
                        nd = new NotifyDescriptor.Message(
                            msg,
                            NotifyDescriptor.INFORMATION_MESSAGE
                        );
                        DialogDisplayer.getDefault().notify(nd);
                    }
                }
            }
        }
    }
}

```

```

        } catch (HttpException ex) {
            output("502 Proxy Error: endpoint not available.");
        } finally {
            executor.shutdown();
            this.setCursor(Cursor.getDefaultCursor());
        }

    }

    } else {
        resetEntitiesAndDisplay();
        Tab.setSelectedIndex(TAB_VISUAL);
    }
}

private void DeleteButtonActionPerformed(java.awt.event.ActionEvent evt)
{
    int row = QueriesTable.getSelectedRow();
    if(row != -1) {
        String queryName;
        queryName = (String) qtb.getValueAt(row,
qtb.findColumn("Name"));
        coll.getEntitiesCollection().removeQuery(queryName);
        qtb.deleteSource(row);
        entityModelChange();
        queryModelChange();
        timeLinesCollection.createFilters(etb.getFlattenedCollection());
        FirstFilterComboBox.setModel(
            timeLinesCollection.getFilterComboBoxModel()
        );
        FirstFilterComboBox.setSelectedIndex(0);
        timeLinesCollection.createTimeLines(
            etb.getFlattenedCollection(),
            (String) FirstFilterComboBox.getSelectedItem()
        );
        paintVisualDisplay();
    }
}

private void TitleTextBoxKeyReleased(java.awt.event.KeyEvent evt) {
    if (evt.getKeyCode() == KeyEvent.VK_ENTER) {
        coll.setName(TitleTextBox.getText());
        setName(coll.getName());
        NotesTextBox.requestFocus();
    }
}

private void HasNameCheckBoxActionPerformed(java.awt.event.ActionEvent
evt) {
    if (HasNameCheckBox.isSelected()) {
        NameTextField.setEnabled(true);
    } else {
        NameTextField.setEnabled(false);
    }
}

private void
HasIdentifierCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {
    if (HasIdentifierCheckBox.isSelected()) {
        IdentifierTextField.setEnabled(true);
    }
}

```

```

        } else {
            IdentifierTextField.setEnabled(false);
        }
    }

    private void
    HasDepictionCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {
        if (HasDepictionCheckBox.isSelected()) {
            DepictionTextField.setEnabled(true);
        } else {
            DepictionTextField.setEnabled(false);
        }
    }

    private void
    HasConsistsCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {
        if (HasConsistsCheckBox.isSelected()) {
            MaterialTextField.setEnabled(true);
        } else {
            MaterialTextField.setEnabled(false);
        }
    }

    private void HasTypeCheckBoxActionPerformed(java.awt.event.ActionEvent
    evt) {
        if (HasTypeCheckBox.isSelected()) {
            TypeTextField.setEnabled(true);
        } else {
            TypeTextField.setEnabled(false);
        }
    }

    private void
    HasTechniqueCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {
        if (HasTechniqueCheckBox.isSelected()) {
            TechniqueTextField.setEnabled(true);
        } else {
            TechniqueTextField.setEnabled(false);
        }
    }

    private void Group1ComboBoxActionPerformed(java.awt.event.ActionEvent
    evt) {
        String selectedItem = (String) Group1ComboBox.getSelectedItem();
        if (selectedItem.equalsIgnoreCase("None")) {
            runFirstFilter();
            setComboBoxes(
                true,
                false,
                false,
                false
            );
        } else {
            setFilter1();
            runSubGroup1();
        }
    }

    private void
    FirstFilterComboBoxActionPerformed(java.awt.event.ActionEvent evt) {
        String selectedItem = (String)

```

```

FirstFilterComboBox.getSelectedItem();
    runFirstFilter();
    if (selectedItem.equalsIgnoreCase("None")) {
        setComboBoxes(
            false,
            false,
            false,
            false
        );
    } else {
        setSubGroup1();
    }
}

private void
Group1FilterComboBoxActionPerformed(java.awt.event.ActionEvent evt) {
    String selectedItem = (String)
Group1FilterComboBox.getSelectedItem();
    runSubGroup1();
    if (selectedItem.equalsIgnoreCase("None")) {
        setComboBoxes(
            true,
            true,
            false,
            false
        );
    } else {
        setSubGroup2();
    }
}

private void
Group2FilterComboBoxActionPerformed(java.awt.event.ActionEvent evt) {
    runSubGroup2();
}

private void Group2ComboBoxActionPerformed(java.awt.event.ActionEvent
evt) {
    String selectedItem = (String) Group2ComboBox.getSelectedItem();
    if (selectedItem.equalsIgnoreCase("None")) {
        runSubGroup1();
        setComboBoxes(
            true,
            true,
            true,
            false
        );
    } else {
        setFilter2();
        runSubGroup2();
    }
}

private void
CreatedByCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {
    if (CreatedByCheckBox.isSelected()) {
        CreatedByTextField.setEnabled(true);
    } else {
        CreatedByTextField.setEnabled(false);
    }
}
}

```



```

private void SaveButtonActionPerformed(java.awt.event.ActionEvent evt)
{
    coll.setNotes(NotesTextBox.getText());
    CollectionXMLWriter writer;
    try {
        writer = new CollectionXMLWriterImpl(
            coll,
            etb.getFlattenedCollection()
        );
        writer.build();
        writer.print();
        String msg = "The collection has been saved.";
        NotifyDescriptor nd;
        nd = new NotifyDescriptor.Message(
            msg,
            NotifyDescriptor.INFORMATION_MESSAGE
        );
        DialogDisplayer.getDefault().notify(nd);

    } catch (ParserConfigurationException ex) {
        Exceptions.printStackTrace(ex);
    }
}

private void HasLimitCheckBoxActionPerformed(java.awt.event.ActionEvent
evt) {
    if (HasLimitCheckBox.isSelected()) {
        LimitTextField.setEnabled(true);
    } else {
        LimitTextField.setEnabled(false);
    }
}

// Variables declaration - do not modify
private javax.swing.JLabel AndText1;
private javax.swing.JScrollPane CollectionDisplayScrollPane;
private javax.swing.JButton CreateButton;
private javax.swing.JCheckBox CreatedByCheckBox;
private javax.swing.JTextField CreatedByTextField;
private javax.swing.JTextField CreationEndYearTextField;
private javax.swing.JTextField CreationStartYearTextField;
private javax.swing.JLabel CreationYearLabel;
private javax.swing.JButton DeleteButton;
private javax.swing.JTextField DepictionTextField;
private javax.swing.JPanel Entities;
private javax.swing.JScrollPane EntitiesScrollPane;
private javax.swing.JTable EntitiesTable;
private javax.swing.JLabel ExistingQueriesText;
private javax.swing.JComboBox<String> FirstFilterComboBox;
private javax.swing.JLabel FirstFilterLabel;
private javax.swing.JComboBox<String> Group1ComboBox;
private javax.swing.JComboBox<String> Group1FilterComboBox;
private javax.swing.JLabel Group1FilterLabel;
private javax.swing.JLabel Group1Label;
private javax.swing.JComboBox<String> Group2ComboBox;
private javax.swing.JComboBox<String> Group2FilterComboBox;
private javax.swing.JLabel Group2FilterLabel;
private javax.swing.JLabel Group2Label;
private javax.swing.JCheckBox HasConsistsCheckBox;
private javax.swing.JCheckBox HasDepictionCheckBox;

```

```

private javax.swing.JCheckBox HasIdentifierCheckBox;
private javax.swing.JCheckBox HasImageCheckBox;
private javax.swing.JCheckBox HasLimitCheckBox;
private javax.swing.JCheckBox HasNameCheckBox;
private javax.swing.JCheckBox HasTechniqueCheckBox;
private javax.swing.JCheckBox HasTypeCheckBox;
private javax.swing.JTextField IdentifierTextField;
private javax.swing.JPanel Information;
private javax.swing.JTextField LimitTextField;
private javax.swing.JTextField MaterialTextField;
private javax.swing.JTextField NameTextField;
private javax.swing.JLabel NotesText;
private javax.swing.JTextArea NotesTextBox;
private javax.swing.JScrollPane NotesTextPanel;
private javax.swing.JPanel Queries;
private javax.swing.JScrollPane QueriesScrollPane;
private javax.swing.JTable QueriesTable;
private javax.swing.JLabel QueryBuilderText;
private javax.swing.JTextField QueryNameTextField;
private javax.swing.JSeparator QuerySeparator;
private javax.swing.JButton ResetButton;
private javax.swing.JButton RunButton;
private javax.swing.JButton SaveButton;
private javax.swing.JComboBox<String> SourceComboBox;
private javax.swing.JLabel SourceNameTextLabel;
private javax.swing.JLabel SourceTextLabel;
private javax.swing.JTabbedPane Tab;
private javax.swing.JTextField TechniqueTextField;
private javax.swing.JTextField TitleTextBox;
private javax.swing.JTextField TypeTextField;
private javax.swing.JSeparator VerticalSeparator;
private javax.swing.JPanel Visualizer;
private javax.swing.JLabel ZoomInLabel;
private javax.swing.JLabel ZoomOutLabel;
private javax.swing.JSlider ZoomSlider;
private
org.jghill.timelinesvisualizercollections.display.CollectionDisplayPanel
collectionDisplayPanel;
private javax.swing.JSeparator jSeparator1;
// End of variables declaration

private final static String ROW_LIMIT = "200";

private final EntityTableModel etb;
private final QueryTableModel qtb;
private final Collection coll;
private final InstanceContent content = new InstanceContent();
private final AbstractLookup abLookup = new AbstractLookup(content);
private final Lookup.Result <Source> sources;

private final TimeLineCollection timeLinesCollection;

@Override
public void componentOpened() {
    setName(coll.getName());
    associateLookup(abLookup);
    queryModelChange();

    if (etb.getFlattenedCollection().length > 0) {
        setFirstFilter();
        runFirstFilter();
    }
}

```

```

        paintVisualDisplay();
    }
}

@Override
public void componentClosed() {}

void writeProperties(java.util.Properties p) {
    p.setProperty("version", "1.0");
}

void readProperties(java.util.Properties p) {
    String version = p.getProperty("version");
}

/**
 * A getter for the collection to be returned from this top component.
 *
 * @return the collection.
 */
public Collection getCollection() {
    return coll;
}

/**
 * Resets the query building section of the GUI.
 */
private void resetQueryBuilder() {

    SourceNameTextLabel.setText("");

    SourceComboBox.setSelectedIndex(0);
    QueryNameTextField.setText("");

    CreationEndYearTextField.setText("");
    CreationStartYearTextField.setText("");

    HasNameCheckBox.setSelected(false);
    NameTextField.setText("");
    NameTextField.setEnabled(false);

    HasIdentifierCheckBox.setSelected(false);
    IdentifierTextField.setText("");
    IdentifierTextField.setEnabled(false);

    HasDepictionCheckBox.setSelected(false);
    DepictionTextField.setText("");
    DepictionTextField.setEnabled(false);

    HasConsistsCheckBox.setSelected(false);
    MaterialTextField.setText("");
    MaterialTextField.setEnabled(false);

    HasTypeCheckBox.setSelected(false);
    TypeTextField.setText("");
    TypeTextField.setEnabled(false);

    HasTechniqueCheckBox.setSelected(false);
    TechniqueTextField.setText("");
    TechniqueTextField.setEnabled(false);
}

```

```

        CreatedByCheckBox.setSelected(false);
        CreatedByTextField.setText("");
        CreatedByTextField.setEnabled(false);

        HasLimitCheckBox.setSelected(true);
        LimitTextField.setText(ROW_LIMIT);
        LimitTextField.setEnabled(true);

        HasImageCheckBox.setSelected(true);
    }

    /**
     * Checks whether the query builder section has been filled correctly.
     *
     * @return where it has been filled correctly.
     */
    private boolean querySettingsAreValid() {
        if
        (SourceComboBox.getSelectedItem().toString().equalsIgnoreCase("Select . . .
        ")) {
            return false;
        }

        String startYear = CreationStartYearTextField.getText();
        if(startYear.length() > 4) {return false;}
        if(!Pattern.matches("[0-9]+", startYear) && !startYear.isEmpty())
        {return false;}

        String endYear = CreationEndYearTextField.getText();
        if(endYear.length() > 4) {return false;}
        if(!Pattern.matches("[0-9]+", endYear) && !endYear.isEmpty())
        {return false;}

        if(QueryNameTextField.getText().isEmpty()) {return false;}
        if(SourceComboBox.getSelectedItem() == null) {return false;}
        if(!LimitTextField.getText().matches("^-?\\d+$")) {return false;}
        if(Integer.parseInt(LimitTextField.getText()) < 0) {return false;}
        if(Integer.parseInt(LimitTextField.getText()) > 1000) {return
        false;}

        return true;
    }

    /**
     * Creates a query.
     */
    private void createQuery() {
        QuerySettings settings;
        settings = new QuerySettings(

            SourceCollection.getSource((String)
SourceComboBox.getSelectedItem()),
            QueryNameTextField.getText().trim(),

            CreationStartYearTextField.getText().trim().toLowerCase(),
            CreationEndYearTextField.getText().trim().toLowerCase(),

            HasNameCheckBox.isSelected(),

```

```

        NameTextField.getText().trim().toLowerCase(),

        HasIdentifierCheckBox.isSelected(),
        IdentifierTextField.getText().trim().toLowerCase(),

        HasDepictionCheckBox.isSelected(),
        DepictionTextField.getText().trim().toLowerCase(),

        HasConsistsCheckBox.isSelected(),
        MaterialTextField.getText().trim().toLowerCase(),

        HasTypeCheckBox.isSelected(),
        TypeTextField.getText().trim().toLowerCase(),

        HasTechniqueCheckBox.isSelected(),
        TechniqueTextField.getText().trim().toLowerCase(),

        CreatedByCheckBox.isSelected(),
        CreatedByTextField.getText().trim().toLowerCase(),

        HasLimitCheckBox.isSelected(),
        LimitTextField.getText().trim().toLowerCase(),

        HasImageCheckBox.isSelected()

    );
    QueryShell qry;
    QueryBuilder builder = QueryBuilder.getInstance();
    qry = builder.buildQuery(settings);
    coll.getQueriesCollection().addQuery(qry);
    resetQueryBuilder();
    queryModelChange();
}

/**
 * Fires changes to the query model table;
 */
private void queryModelChange() {
    qtb.fireTableDataChanged();
}

/**
 * Fires changes to the entity model table;
 */
private void entityModelChange() {
    etb.fireTableDataChanged();
}

/**
 * Outputs an explanation of the action.
 *
 * @param text toString of the returned entity.
 */
private void output(String text) {
    InputOutput io = IOProvider.getDefault().getIO("Main", false);
    io.getOut().println(text);
}

/**
 * Resets the entities collection and the collection display.
 */

```

```

private void resetEntitiesAndDisplay() {
    etb.clearAll();
    collectionDisplayPanel.clear();
}

/**
 * Paints the visual display.
 */
private void paintVisualDisplay() {
    if (timeLinesCollection.getCount() > 0) {
        collectionDisplayPanel.setArray(timeLinesCollection);
        Tab.setSelectedIndex(TAB_VISUAL);
    } else {
        collectionDisplayPanel.clear();
    }
}

@Override
public void focusGained(FocusEvent e) {
    EntityDisplay temp = (EntityDisplay) e.getSource();

    content.add(temp.getLookup().lookupResult(Entities.class).allInstances().iterator().next());
}

@Override
public void focusLost(FocusEvent e) {
    EntityDisplay temp = (EntityDisplay) e.getSource();

    content.remove(temp.getLookup().lookupResult(Entities.class).allInstances().iterator().next());
}

/**
 * A getter for the First Filter combobox.
 *
 * @return the First Filter comboBox.
 */
public JComboBox<String> getFirstFilter() {
    return FirstFilterComboBox;
}

/**
 * Sets the status of the combo boxes.
 *
 * @param combol
 * @param filter1
 * @param combo2
 * @param filter2
 */
private void setComboBoxes(
    boolean combol,
    boolean filter1,
    boolean combo2,
    boolean filter2
) {
    Group1ComboBox.setEnabled(combol);
    Group1FilterComboBox.setEnabled(filter1);
    Group2ComboBox.setEnabled(combo2);
    Group2FilterComboBox.setEnabled(filter2);
}

```

```

private ManMadeObject[] filter1;
private ManMadeObject[] group1;
private ManMadeObject[] group2;

/**
 * Sets the First Filter.
 */
private void setFirstFilter() {
    timeLinesCollection.createFilters(etb.getFlattenedCollection());
    FirstFilterComboBox.setModel(
        timeLinesCollection.getFilterComboBoxModel()
    );
}

FirstFilterComboBox.setSelectedItem(timeLinesCollection.chooseFilter());
if (FirstFilterComboBox.getSelectedItem().equals("None")) {
    setComboBoxes(
        false,
        false,
        false,
        false
    );
} else {
    setComboBoxes(
        true,
        false,
        false,
        false
    );
}
}

/**
 * Runs the First Filter.
 */
private void runFirstFilter() {
    timeLinesCollection.createTimeLines(
        etb.getFlattenedCollection(),
        (String) FirstFilterComboBox.getSelectedItem()
    );
    filter1 = timeLinesCollection.getAllTimeLineObjects();
    group1 = null;
    group2 = null;
    paintVisualDisplay();
}

/**
 * Sets the first sub group model.
 */
private void setSubGroup1() {
    Group1ComboBox.setModel(
        timeLinesCollection.getCategoryComboBoxModel()
    );
    setComboBoxes(
        true,
        false,
        false,
        false
    );
}
}

```

```

/**
 * Sets the first sub group.
 */
private void runSubGroup1() {
    runFirstFilter();
    String category = (String) Group1ComboBox.getSelectedItem();
    group1 = timeLinesCollection.getTimeLineObjects(category);
    timeLinesCollection.createTimeLines(
        group1,
        (String) Group1FilterComboBox.getSelectedItem()
    );
    group2 = null;
    paintVisualDisplay();
}

/**
 * Sets filter for group 1.
 */
private void setFilter1() {
    timeLinesCollection.createFilters(filter1);
    Group1FilterComboBox.setModel(
        timeLinesCollection.getFilterComboBoxModel()
    );
}

Group1FilterComboBox.setSelectedItem(timeLinesCollection.chooseFilter());
setComboBoxes(
    true,
    true,
    false,
    false
);
}

/**
 * Sets the filter for group 2.
 */
private void setSubGroup2() {
    Group2ComboBox.setModel(
        timeLinesCollection.getCategoryComboBoxModel()
    );
    setComboBoxes(
        true,
        true,
        true,
        false
    );
}

/**
 * Sets the second sub group.
 */
private void runSubGroup2() {
    runSubGroup1();
    String category = (String) Group2ComboBox.getSelectedItem();
    group2 = timeLinesCollection.getTimeLineObjects(category);
    timeLinesCollection.createTimeLines(
        group2,
        (String) Group2FilterComboBox.getSelectedItem()
    );
    paintVisualDisplay();
    setComboBoxes(

```



```

        true,
        true,
        true,
        true
    );
}

/**
 * Sets the filter for sub group 2.
 */
private void setFilter2() {
    timeLinesCollection.createFilters(group1);
    Group2FilterComboBox.setModel(
        timeLinesCollection.getFilterComboBoxModel()
    );

    Group2FilterComboBox.setSelectedItem(timeLinesCollection.chooseFilter());
}
}

```

12.1.9 CollectionXMLParserImpl.java

```
package org.jghill.timelinesvisualizercollectionxml;

import java.io.File;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;
import org.jghill.timelinesvisualizercollections.Collection;
import org.jghill.timelinesvisualizercollections.CollectionImpl;
import org.jghill.timelinevisualizerentities.Entities;
import org.jghill.timelinevisualizerentities.ManMadeObject;
import org.jghill.timelinevisualizerentitiescollection.EntitiesCollection;
import org.jghill.timelinevisualizerqueries.QueryShell;
import org.jghill.timelinevisualizerqueries.SPARQLQueryShell;
import org.jghill.timelinevisualizerqueriescollection.QueriesCollection;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;

/**
 * An implementation of the Collection parser.
 *
 * @author JGHill
 */
public class CollectionXMLParserImpl implements CollectionXMLParser {

    private final Document doc;
    private final DocumentBuilder builder;
    private final XPath path;

    /**
     * Constructor.
     *
     * @param f the file to be passed.
     * @throws ParserConfigurationException
     * @throws SAXException
     * @throws IOException
     */
    public CollectionXMLParserImpl(File f) throws
    ParserConfigurationException, SAXException, IOException {
        DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
        builder = factory.newDocumentBuilder();
        doc = builder.parse(f);
        XPathFactory xFactory = XPathFactory.newInstance();
        path = xFactory.newXPath();
    }

    @Override
    public Collection parseCollection() throws XPathExpressionException {

        // Setup basic collection info.
        String name;
        String notes;
        name = path.evaluate("/collection/information/name", doc);
        notes = path.evaluate("/collection/information/notes", doc);
    }
}
```

```

        // Setup the Queries.
        QueriesCollection queries = new QueriesCollection();
        int queriesCount;
        queriesCount =
Integer.parseInt(path.evaluate("count(/collection/queries/query)", doc));
        for (int i = 1; i <= queriesCount; i++) {
            String queryName = path.evaluate("/collection/queries/query[" +
i + "]/name", doc);
            String queryType = path.evaluate("/collection/queries/query[" +
i + "]/type", doc);
            if (queryType.equalsIgnoreCase("sparql endpoint")) {
                String queryString =
path.evaluate("/collection/queries/query[" + i + "]/querystring", doc);
                String address = path.evaluate("/collection/queries/query["
+ i + "]/address", doc);
                String cidoc = path.evaluate("/collection/queries/query[" +
i + "]/cidoc", doc);
                int limit =
Integer.parseInt(path.evaluate("/collection/queries/query[" + i + "]/limit",
doc));

                QueryShell query = new SPARQLQueryShell(
                    queryString,
                    address,
                    cidoc,
                    queryName,
                    limit
                );
                queries.addQuery(query);
            }
        }

        // Setup the Entities.
        EntitiesCollection entities = new EntitiesCollection("Entities");
        int entitiesCount;
        entitiesCount =
Integer.parseInt(path.evaluate("count(/collection/entities/entity)", doc));
        for (int i = 1; i <= entitiesCount; i++) {
            String identifier =
path.evaluate("/collection/entities/entity[" + i + "]/identifier", doc);
            String entityName =
path.evaluate("/collection/entities/entity[" + i + "]/name", doc);
            String query = path.evaluate("/collection/entities/entity[" + i
+ "]/query", doc);
            String source = path.evaluate("/collection/entities/entity[" +
i + "]/source", doc);
            String consists = path.evaluate("/collection/entities/entity["
+ i + "]/consists", doc);
            String creator = path.evaluate("/collection/entities/entity[" +
i + "]/creator", doc);
            String curatorial =
path.evaluate("/collection/entities/entity[" + i + "]/curatorial", doc);
            String depicts = path.evaluate("/collection/entities/entity[" +
i + "]/depicts", doc);
            String description =
path.evaluate("/collection/entities/entity[" + i + "]/description", doc);
            String image = path.evaluate("/collection/entities/entity[" + i
+ "]/image", doc);
            String object = path.evaluate("/collection/entities/entity[" +
i + "]/object", doc);
            String technique = path.evaluate("/collection/entities/entity["

```

```

+ i + "]/technique", doc);
    String type = path.evaluate("/collection/entities/entity[" + i
+ "]/type", doc);
    String year = path.evaluate("/collection/entities/entity[" + i
+ "]/year", doc);
    Entities entity = new ManMadeObject(
        entityName,
        identifier,
        source,
        query,
        depicts,
        consists,
        type,
        technique,
        image,
        year,
        creator,
        object,
        description,
        curatorial
    );
    entities.addThing(entity);
}

// Creat the collection.
Collection collection = new CollectionImpl(
    name,
    entities,
    queries
);
collection.setNotes(notes);
return collection;
}
}

```

12.1.10 CollectionXMLWriterImpl.java

```
package org.jghill.timelinesvisualizercollectionxml;

import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.jghill.timelinesvisualizercollections.Collection;
import org.jghill.timelinevisualizerentities.Entities;
import org.jghill.timelinevisualizerentities.ManMadeObject;
import org.jghill.timelinevisualizerentities.PhysicalThing;
import org.jghill.timelinevisualizerqueries.QueryShell;
import org.jghill.timelinevisualizerqueries.SPARQLQueryShell;
import org.jghill.timelinevisualizerqueriescollection.QueriesCollection;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Text;

/**
 * An implementation of the collection writer.
 *
 * @author JGHill
 */
public class CollectionXMLWriterImpl implements CollectionXMLWriter {

    private final Collection collection;
    private final ManMadeObject[] entities;
    private final DocumentBuilder builder;
    private Document doc;

    /**
     * This is a constructor.
     *
     * @param collection to be written to XML.
     * @param entities to be written.
     * @throws javax.xml.parsers.ParserConfigurationException
     */
    public CollectionXMLWriterImpl(
        Collection collection,
        ManMadeObject[] entities
    ) throws ParserConfigurationException {
        this.collection = collection;
        this.entities = entities;
        builder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
    }

    @Override
    public void build() {
        doc = builder.newDocument();
        Element rootElement = doc.createElement("collection");
        doc.appendChild(rootElement);
        createCollection(rootElement);
    }
}
```

```

@Override
public void print() {
    Transformer trans;
    try {
        trans = TransformerFactory.newInstance().newTransformer();
        trans.setOutputProperty(OutputKeys.INDENT, "yes");
        trans.setOutputProperty("{http://xml.apache.org/xslt}indent-
amount", "2");
        DOMSource source = new DOMSource(doc);
        StreamResult file = new StreamResult(
            new File(
                "Data/Collections/" + collection.getName() +
                ".xml"
            )
        );
        trans.transform(source, file);
    } catch (TransformerException ex) {
        System.out.println("Transformer " + ex);
    }
}

/**
 * Creates a textual Element.
 *
 * @param name for the element.
 * @param text to be contained with the element.
 * @return the element.
 */
private Element createTextElement(
    String name,
    String text
) {
    Text t = doc.createTextNode(text);
    Element e = doc.createElement(name);
    e.appendChild(t);
    return e;
}

/**
 * Creates a collection of entities.
 *
 * @param root element.
 */
private void createCollection(Element root) {
    root.appendChild(createInfo());
    root.appendChild(createQueries());
    root.appendChild(createEntities());
}

/**
 * Creates a node for the information about the object.
 *
 * @return a node containing the info.
 */
private Element createInfo() {
    Element e = doc.createElement("information");
    e.appendChild(createTextElement("name", "" + collection.getName()));
    e.appendChild(createTextElement("notes", "" +
collection.getNotes()));
    return e;
}

```

```

    }

    /**
     * Creates a node for containing the queries associated with the object.
     *
     * @return a node of the queries.
     */
    private Element createQueries() {
        Element e = doc.createElement("queries");
        QueriesCollection queries = collection.getQueriesCollection();
        for (QueryShell query : queries) {
            Element q = doc.createElement("query");
            q.appendChild(createTextElement("name", "" +
query.getQueryName()));
            q.appendChild(createTextElement("type", "" +
query.getQueryType()));
            if (query.getQueryType().equalsIgnoreCase("sparql endpoint")) {
                SPARQLQueryShell sparql = (SPARQLQueryShell) query;
                q.appendChild(createTextElement("querystring", "" +
sparql.getQueryString()));
                q.appendChild(createTextElement("address", "" +
sparql.getServiceAddress()));
                q.appendChild(createTextElement("cidoc", "" +
sparql.getCIDOCAddress()));
                q.appendChild(createTextElement("limit", "" +
sparql.getLimit()));
            }
            e.appendChild(q);
        }
        return e;
    }

    /**
     * Creates a node representing entities associated with the collection.
     *
     * @return a node of the Entities.
     */
    private Element createEntities() {
        Element e = doc.createElement("entities");
        for (Entities entity : entities) {
            Element en = doc.createElement("entity");
            en.appendChild(createTextElement("identifier", "" +
entity.getIdentifier()));
            en.appendChild(createTextElement("name", "" +
entity.getName()));
            en.appendChild(createTextElement("query", "" +
entity.getQueryName()));
            en.appendChild(createTextElement("source", "" +
entity.getSourceName()));
            if (entity instanceof PhysicalThing) {
                PhysicalThing thing = (PhysicalThing) entity;
                en.appendChild(createTextElement("consists", "" +
thing.getConsists()));
                en.appendChild(createTextElement("creator", "" +
thing.getCreator()));
                en.appendChild(createTextElement("curatorial", "" +
thing.getCuratorial()));
                en.appendChild(createTextElement("depicts", "" +
thing.getDepicts()));
                en.appendChild(createTextElement("description", "" +
thing.getDescription()));
            }
        }
    }

```

```

        if (thing.getImageURL() != null) {
            en.appendChild(createTextElement("image", "" +
thing.getImageURL().toString()));
        }
        en.appendChild(createTextElement("object", "" +
thing.getObject()));
        en.appendChild(createTextElement("technique", "" +
thing.getTechnique()));
        en.appendChild(createTextElement("type", "" +
thing.getType()));
        if (thing.getTimeSpan() != null) {
            en.appendChild(createTextElement("year", "" +
thing.getTimeSpan().toString()));
        }
    }
    e.appendChild(en);
}
return e;
}
}

```


12.2 Dispatcher Module

This section contains all Java code from the Dispatcher module.

12.2.1 Dispatcher.java

```
package org.jghill.timelinesvisualizerdispatcher;

import java.util.Date;
import java.util.concurrent.Callable;
import org.jghill.timelinevisualizerentitiescollection.EntitiesCollection;
import org.jghill.timelinevisualizerqueries.QueryShell;
import org.jghill.timelinevisualizerqueriescollection.QueriesCollection;
import org.netbeans.api.io.IOProvider;
import org.netbeans.api.io.InputOutput;

/**
 * A dispatcher sends queries and collects the results to create a
 * collection
 * of entities.
 *
 * @author JGHill
 */
public class Dispatcher implements Callable {

    private final QueriesCollection queries;

    /**
     * The Constructor.
     *
     * @param queries to be dispatched to endpoints.
     */
    public Dispatcher(QueriesCollection queries) {
        this.queries = queries;
    }

    /**
     * Runs all queries in a collection.
     *
     * @return a collection of entities built from the queries.
     */
    @Override
    public EntitiesCollection call() {

        EntitiesCollection entities;
        entities = new EntitiesCollection("Collection");

        output("Running campaigns");
        output("");
        for(QueryShell q : queries) {
            output("Running query: " + q.getQueryName());
            EntitiesCollection queryResults;
            queryResults = q.run();
            if (queryResults != null) {
                entities.addThing(queryResults);
            }
            q.setLastRunDate(new Date());
        }
    }
}
```

```

        }
        output("");
        output("All campaigns run");
        output("");

        return entities;
    }

    /**
     * Outputs an explanation of the action.
     *
     * @param text toString of the returned entity.
     */
    private void output(String text) {
        InputOutput io = IOProvider.getDefault().getIO("Main", false);
        io.getOut().println(text);
    }
}

```

12.3 Entities Module

This section contains all Java code from the Entities module.

12.3.1 Entities.java

```
package org.jghill.timelinevisualizerentities;

import java.util.Objects;

/**
 * A class for representing collections & physical things as components of
 * a
 * composition.
 *
 * @author JGHill
 */
abstract public class Entities implements Comparable {

    private final String name;
    private final String identifier;
    private final String source;
    private final String query;

    /**
     * Constructor.
     *
     * @param name of the Entity.
     * @param identifier for the Entity.
     * @param source of the Entity.
     * @param query from which the Entity was pulled.
     */
    public Entities(
        String name,
        String identifier,
        String source,
        String query
    ) {
        this.name = name;
        this.identifier = identifier;
        this.source = source;
        this.query = query;
    }

    /**
     * Returns the name of the entity.
     *
     * @return the name.
     */
    public String getName() {
        return name;
    }

    /**
     * Returns the identifier that the institution applied to the entity.
     *
     * @return the identifier.
     */
}
```

```

    */
    public String getIdentifier() {
        return identifier;
    }

    /**
     * Returns the Source name from which the entity was obtained.
     *
     * @return the Source name.
     */
    public String getSourceName() {
        return source;
    }

    /**
     * Returns the Query name which the entity was returned from.
     *
     * @return the Query name.
     */
    public String getQueryName() {
        return query;
    }

    @Override
    public boolean equals(Object o) {
        if(o.getClass() == this.getClass()) {
            Entities e = (Entities) o;
            return identifier.equals(e.identifier);
        } else {
            return false;
        }
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 89 * hash + Objects.hashCode(this.identifier);
        return hash;
    }

    @Override
    public int compareTo(Object o) {
        Entities e = (Entities) o;
        return name.compareTo(e.name);
    }
}

```

12.3.2 ManMadeObject.java

```
package org.jghill.timelinevisualizerentities;

/**
 * A class representing objects that have been designed & created by humans.
 *
 * @author JGHill
 */
public class ManMadeObject extends PhysicalThing {

    /**
     * Constructor for the class.
     *
     * @param name of the object.
     * @param identifier to recognise the object.
     * @param source of the object.
     * @param query that collected the object.
     * @param depicts some visual entity.
     * @param consists of some material.
     * @param type of the object.
     * @param technique used to produce the object.
     * @param image representing the object.
     * @param year the object is estimated to have been produced.
     * @param creator of the object.
     * @param object address for identification online.
     * @param description of the object.
     * @param curatorial comments about the object.
     */
    public ManMadeObject(
        String name,
        String identifier,
        String source,
        String query,
        String depicts,
        String consists,
        String type,
        String technique,
        String image,
        String year,
        String creator,
        String object,
        String description,
        String curatorial
    ) {
        super(
            name,
            identifier,
            source,
            query,
            depicts,
            consists,
            type,
            technique,
            image,
            year,
            creator,
            object,
            description,
            curatorial
        );
    }
}
```

```
    }  
    );  
}
```

12.3.3 PhysicalThing.java

```
package org.jghill.timelinevisualizerentities;

import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import javax.imageio.ImageIO;

/**
 * An abstraction of a physical object for subclasses that represent real
 * objects.
 *
 * @author JamesGHill
 */
public abstract class PhysicalThing extends Entities {

    private static final int YEAR_LENGTH = 4;

    private final Integer timeSpan;

    private final String depicts;
    private final String consists;
    private final String type;
    private final String technique;
    private final String creator;
    private final String object;

    private final String description;
    private final String curatorial;

    protected URL imageURL;
    protected File imageFile;

    /**
     * Constructor.
     *
     * @param name
     * @param identifier
     * @param source
     * @param query
     * @param depicts
     * @param consists
     * @param type
     * @param technique
     * @param image
     * @param beginYear
     * @param creator
     * @param object
     * @param description
     * @param curatorial
     */
    public PhysicalThing(
        String name,
        String identifier,
        String source,
```

```

        String query,
        String depicts,
        String consists,
        String type,
        String technique,
        String image,
        String beginYear,
        String creator,
        String object,
        String description,
        String curatorial
    ) {
        super(
            name,
            identifier,
            source,
            query
        );
        this.depicts = depicts;
        this.consists = consists;
        this.type = type;
        this.technique = technique;
        this.creator = creator;
        this.object = object;
        this.description = description;
        this.curatorial = curatorial;
        this.imageURL = createURL(image);
        this.timeSpan = createYear(beginYear);
        checkForImage();
    }

    /**
     * Check that image file exists already and assign if it does.
     */
    private void checkForImage() {
        if (imageFile == null) {
            File temp;
            temp = new File("Data/Images/" + super.getIdentifier() +
".jpg");
            if (temp.exists()) {
                imageFile = temp;
            }
        }
    }

    /**
     * Creates a URL from the returned String.
     *
     * @param text the URL text from the query results.
     * @return a URL object.
     */
    private URL createURL(String text) {
        URL temp = null;
        try {
            if (!text.isEmpty()) {
                temp = new URL(text);
            }
        } catch (MalformedURLException ex) {
            temp = null;
        }
        return temp;
    }

```



```

    }

    /**
     * Creates a year as int from a 4 character String or returns a -1.
     *
     * @param yearText the String to be transformed.
     * @return the year or a -1.
     */
    private Integer createYear(String yearText) {
        if (yearStringIsInt(yearText)) {
            return Integer.parseInt(yearText);
        } else {
            return extractYearFromString(yearText);
        }
    }

    /**
     * Checks if a string is a year.
     *
     * @param yearText the string to check.
     * @return true if the text can convert to a year.
     */
    private boolean yearStringIsInt(String yearText) {
        if (yearText.length() == 4) {
            for(int i = 0; i < yearText.length(); i++) {
                if (!Character.isDigit(yearText.charAt(i))) {
                    return false;
                }
            }
            return true;
        } else {
            return false;
        }
    }

    /**
     * Attempts to extract a 4-digit year from a String; otherwise returns
a    * null.
     *
     * @param yearText the String to extract from.
     * @return an integer year.
     */
    private Integer extractYearFromString(String yearText) {
        int digitCount = 0;
        for(int i = 0; i < yearText.length(); i++) {
            if (Character.isDigit(yearText.charAt(i))) {
                digitCount++;
                if (digitCount == YEAR_LENGTH) {
                    int j = i + 1;
                    return Integer.parseInt(yearText.substring(j -
digitCount, j));
                }
            } else {
                digitCount = 0;
            }
        }
        return null;
    }

    /**

```

```

        * Checks if the image has already been loaded.  If loaded return the
image,
        * otherwise attempt to load the image then return it, otherwise return
        * null.
        *
        * @return the Image related to this Thing or a null.
        */
private BufferedImage getImage() {
    BufferedImage image = null;
    if (imageFile == null) {
        if(imageURL != null) {
            try {
                image = (BufferedImage) ImageIO.read(imageURL);
                File temp;
                temp = new File("Data/Images/" + super.getIdentifier()
+ ".jpg");
                imageFile = temp;
                ImageIO.write(image, "jpg", imageFile);
            } catch(IOException ex) {}
        }
    } else {
        try {
            image = ImageIO.read(imageFile);
        } catch (IOException ex) {}
    }
    return image;
}

/**
 * Creates a thumbnail image from an existing image.
 *
 * @param dimension the dimension the image must fit within.
 * @return the thumbnail image.
 */
public BufferedImage getThumb(int dimension) {
    BufferedImage image = getImage();
    if (image != null) {

        int x = image.getWidth();
        int y = image.getHeight();
        double ratio = ((double) x) / y;

        int w = dimension;
        int h = dimension;

        if(x > dimension || y > dimension) {
            w = (int) Math.min(dimension, ratio * dimension);
            h = (int) Math.min(dimension, dimension / ratio);
        }

        BufferedImage reSize;
        reSize = new BufferedImage(w, h, BufferedImage.TYPE_INT_ARGB);
        Graphics2D g2 = reSize.createGraphics();

        g2.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
RenderingHints.VALUE_INTERPOLATION_BILINEAR);
        g2.drawImage(image, 0, 0, w, h, null);
        g2.dispose();

        return reSize;
    }
}

```

```

        } else {
            return null;
        }
    }

    /**
     * @return the depiction.
     */
    public String getDepicts() {
        return depicts;
    }

    /**
     * @return the material.
     */
    public String getConsists() {
        return consists;
    }

    /**
     * @return the type.
     */
    public String getType() {
        return type;
    }

    /**
     * @return the technique.
     */
    public String getTechnique() {
        return technique;
    }

    /**
     * @return the creator.
     */
    public String getCreator() {
        return creator;
    }

    /**
     * @return the object.
     */
    public String getObject() {
        return object;
    }

    /**
     * @return the description.
     */
    public String getDescription() {
        return description;
    }

    /**
     * @return the curatorial commentary.
     */
    public String getCuratorial() {
        return curatorial;
    }

```

```
/**
 * @return the image source URL.
 */
public URL getImageURL() {
    return imageURL;
}

/**
 * @return the earliest time.
 */
public Integer getTimeSpan() {
    return timeSpan;
}

/**
 * @return the temporary image file.
 */
public File getImageFile() {
    return imageFile;
}
}
```

12.3.4 EntitiesCollection.java

```
package org.jghill.timelinevisualizerentitiescollection;

import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.TreeSet;
import org.jghill.timelinevisualizerentities.Entities;

/**
 * Holds collections of other Entities, whether PhysicalThings or other
 * EntitiesCollections.
 *
 * @author JGHill
 */
public class EntitiesCollection extends Entities {

    private final Set<Entities> list;

    /**
     * Constructor.
     *
     * @param name of this collection of Entities.
     */
    public EntitiesCollection(String name) {
        super(name, "", "", "");
        list = new TreeSet<>();
    }

    /**
     * Allows other entities to be added to this entity.
     *
     * @param e another entity to be added.
     */
    public void addThing(Entities e) {
        list.add(e);
    }

    /**
     * Allows a count of the entities to be returned.
     *
     * @return a count of the entities within the list.
     */
    public int count() {
        return list.size();
    }

    /**
     * Return the entire list.
     *
     * @return the entity list.
     */
    public Set<Entities> getCollectionSet() {
        Set<Entities> arr = new TreeSet<>();
        list.stream().forEach((e) -> {
            if(e instanceof EntitiesCollection) {
                arr.addAll(entitiesFlatten((EntitiesCollection)e));
            } else {
                arr.add(e);
            }
        });
    }
}
```

```

        }
    });
    return arr;
}

/**
 * Flattens the entities collection into a single list.
 *
 * @param coll the EntitiesCollection.
 * @return a list containing all internal non-collections.
 */
private List<Entities> entitiesFlatten(EntitiesCollection coll) {
    List<Entities> arr = new ArrayList<>();
    coll.getCollectionSet().stream().forEach((e) -> {
        if(e instanceof EntitiesCollection) {
            arr.addAll(entitiesFlatten((EntitiesCollection)e));
        } else {
            arr.add(e);
        }
    });
    return arr;
}

/**
 * Remove all Entities which match the passed Query name.
 *
 * @param queryName
 */
public void removeQuery(String queryName) {
    List removable = new ArrayList<>();
    list.stream().forEach((entity) -> {
        if (entity instanceof EntitiesCollection) {
            ((EntitiesCollection) entity).removeQuery(queryName);
        } else {
            if (entity != null) {
                if (entity.getQueryName().equals(queryName)) {
                    removable.add(entity);
                }
            }
        }
    });
    list.removeAll(removable);
}
}

```

12.4 Properties Module

This section contains all Java code from the Properties module.

12.4.1 PropertiesTopComponent.java

```
package org.jghill.timelinevisualizer.properties;

import java.awt.BorderLayout;
import java.awt.image.BufferedImage;
import java.util.Collection;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import org.jghill.timelinevisualizer.entities.ManMadeObject;
import org.netbeans.api.settings.ConvertAsProperties;
import org.openide.awt.ActionID;
import org.openide.util.Lookup;
import org.openide.util.LookupEvent;
import org.openide.util.LookupListener;
import org.openide.windows.TopComponent;
import org.openide.util.NbBundle.Messages;
import org.openide.util.Utilities;

/**
 * A viewer for displaying properties of the currently selected Entity.
 */
@ConvertAsProperties(
    dtd = "-//org.jghill.timelinevisualizer.properties//Properties//EN",
    autostore = false
)
@TopComponent.Description(
    preferredID = "PropertiesTopComponent",
    persistenceType = TopComponent.PERSISTENCE_ALWAYS
)
@TopComponent.Registration(mode = "properties", openAtStartup = true)
@ActionID(category = "Window", id =
    "org.jghill.timelinevisualizer.properties.PropertiesTopComponent")
@TopComponent.OpenActionRegistration(
    displayName = "#CTL_PropertiesAction",
    preferredID = "PropertiesTopComponent"
)
@Messages({
    "CTL_PropertiesAction=Properties",
    "CTL_PropertiesTopComponent=Properties",
    "HINT_PropertiesTopComponent=This is a Properties window"
})
public final class PropertiesTopComponent extends TopComponent implements
    LookupListener {

    /**
     * Constructor.
     */
    public PropertiesTopComponent() {
        initComponents();
        setName(Bundle.CTL_PropertiesTopComponent());
        setToolTipText(Bundle.HINT_PropertiesTopComponent());
    }
}
```

```

        putClientProperty(TopComponent.PROP_CLOSING_DISABLED, Boolean.TRUE);
        putClientProperty(TopComponent.PROP_DRAGGING_DISABLED,
Boolean.TRUE);
        putClientProperty(TopComponent.PROP_MAXIMIZATION_DISABLED,
Boolean.TRUE);
        putClientProperty(TopComponent.PROP_UNDOCKING_DISABLED,
Boolean.TRUE);

        setup();
    }

    /**
     * This method is called from within the constructor to initialize the
form.
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        PropertiesScrollPane = new javax.swing.JScrollPane();
        PropertiesPanel = new javax.swing.JPanel();
        CommentaryLabel = new javax.swing.JLabel();
        CommentaryScrollPane = new javax.swing.JScrollPane();
        CommentaryTextArea = new javax.swing.JTextArea();
        ImagePanel = new javax.swing.JPanel();
        CreationYearTextField = new javax.swing.JTextField();
        CreationYearLabel = new javax.swing.JLabel();
        DepictsLabel = new javax.swing.JLabel();
        DepictsTextField = new javax.swing.JTextField();
        ConsistsTextField = new javax.swing.JTextField();
        ConsistsLabel = new javax.swing.JLabel();
        TypeLabel = new javax.swing.JLabel();
        TypeTextField = new javax.swing.JTextField();
        TechniqueLabel = new javax.swing.JLabel();
        TechniqueTextField = new javax.swing.JTextField();
        IdentifierLabel = new javax.swing.JLabel();
        NameLabel = new javax.swing.JLabel();
        SourceLabel = new javax.swing.JLabel();
        QueryLabel = new javax.swing.JLabel();
        IdentifierTextField = new javax.swing.JTextField();
        SourceTextField = new javax.swing.JTextField();
        QueryTextField = new javax.swing.JTextField();
        DescriptionLabel = new javax.swing.JLabel();
        DescriptionScrollPane = new javax.swing.JScrollPane();
        DescriptionTextArea = new javax.swing.JTextArea();
        NameTextField = new javax.swing.JTextField();
        CreatorLabel = new javax.swing.JLabel();
        CreatorTextField = new javax.swing.JTextField();
        DetailsLabel = new javax.swing.JLabel();
        DetailsTextField = new javax.swing.JTextField();

        setMaximumSize(new java.awt.Dimension(340, 32767));
        setMinimumSize(new java.awt.Dimension(340, 0));
        setOpaque(true);

        PropertiesScrollPane.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneCon
stants.HORIZONTAL_SCROLLBAR_NEVER);

```



```

PropertiesScrollPane.setToolTipText(org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class,
"PropertiesTopComponent.PropertiesScrollPane.toolTipText")); // NOI18N

PropertiesScrollPane.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
    PropertiesScrollPane.setHorizontalScrollBar(null);
    PropertiesScrollPane.setMaximumSize(new java.awt.Dimension(340,
32767));
    PropertiesScrollPane.setMinimumSize(new java.awt.Dimension(340, 0));
    PropertiesScrollPane.setPreferredSize(new java.awt.Dimension(340,
850));

    PropertiesPanel.setMinimumSize(new java.awt.Dimension(0, 0));
    PropertiesPanel.setPreferredSize(new java.awt.Dimension(320, 850));

    org.openide.awt.Mnemonics.setLocalizedText(CommentaryLabel,
org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class,
"PropertiesTopComponent.CommentaryLabel.text")); // NOI18N

CommentaryScrollPane.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

    CommentaryTextArea.setEditable(false);
    CommentaryTextArea.setColumns(20);
    CommentaryTextArea.setFont(new java.awt.Font("Tahoma", 0, 11)); //
NOI18N
    CommentaryTextArea.setLineWrap(true);
    CommentaryTextArea.setRows(6);
    CommentaryTextArea.setWrapStyleWord(true);
    CommentaryScrollPane.setViewportView(CommentaryTextArea);

ImagePanel.setBorder(javax.swing.BorderFactory.createEtchedBorder());
    ImagePanel.setMaximumSize(new java.awt.Dimension(1000, 280));
    ImagePanel.setMinimumSize(new java.awt.Dimension(280, 280));
    ImagePanel.setPreferredSize(new java.awt.Dimension(280, 280));

    javax.swing.GroupLayout ImagePanelLayout = new
javax.swing.GroupLayout(ImagePanel);
    ImagePanel.setLayout(ImagePanelLayout);
    ImagePanelLayout.setHorizontalGroup(

ImagePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 293, Short.MAX_VALUE)
    );
    ImagePanelLayout.setVerticalGroup(

ImagePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 303, Short.MAX_VALUE)
    );

    CreationYearTextField.setEditable(false);
    CreationYearTextField.setBackground(new java.awt.Color(255, 255,
255));

CreationYearTextField.setHorizontalAlignment(javax.swing.JTextField.LEFT);

```

```

CreationYearTextField.setText(org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class, "PropertiesTopComponent.CreationYearTextField.text"));
// NOI18N

    org.openide.awt.Mnemonics.setLocalizedText(CreationYearLabel,
org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class,
"PropertiesTopComponent.CreationYearLabel.text")); // NOI18N

    org.openide.awt.Mnemonics.setLocalizedText(DepictsLabel,
org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class,
"PropertiesTopComponent.DepictsLabel.text")); // NOI18N

    DepictsTextField.setEditable(false);
    DepictsTextField.setBackground(new java.awt.Color(255, 255, 255));

DepictsTextField.setHorizontalAlignment(javax.swing.JTextField.LEFT);

DepictsTextField.setText(org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class, "PropertiesTopComponent.DepictsTextField.text")); //
NOI18N
    DepictsTextField.setMaximumSize(new java.awt.Dimension(196, 20));
    DepictsTextField.setMinimumSize(new java.awt.Dimension(196, 20));
    DepictsTextField.setPreferredSize(new java.awt.Dimension(196, 20));

    ConsistsTextField.setEditable(false);
    ConsistsTextField.setBackground(new java.awt.Color(255, 255, 255));

ConsistsTextField.setHorizontalAlignment(javax.swing.JTextField.LEFT);

ConsistsTextField.setText(org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class, "PropertiesTopComponent.ConsistsTextField.text")); //
NOI18N
    ConsistsTextField.setMaximumSize(new java.awt.Dimension(196, 20));
    ConsistsTextField.setMinimumSize(new java.awt.Dimension(196, 20));
    ConsistsTextField.setPreferredSize(new java.awt.Dimension(196, 20));

    org.openide.awt.Mnemonics.setLocalizedText(ConsistsLabel,
org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class,
"PropertiesTopComponent.ConsistsLabel.text")); // NOI18N

    org.openide.awt.Mnemonics.setLocalizedText(TypeLabel,
org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class,
"PropertiesTopComponent.TypeLabel.text")); // NOI18N

    TypeTextField.setEditable(false);
    TypeTextField.setBackground(new java.awt.Color(255, 255, 255));
    TypeTextField.setHorizontalAlignment(javax.swing.JTextField.LEFT);

TypeTextField.setText(org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class, "PropertiesTopComponent.TypeTextField.text")); // NOI18N
    TypeTextField.setMaximumSize(new java.awt.Dimension(196, 20));
    TypeTextField.setMinimumSize(new java.awt.Dimension(196, 20));
    TypeTextField.setPreferredSize(new java.awt.Dimension(196, 20));

    org.openide.awt.Mnemonics.setLocalizedText(TechniqueLabel,
org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class,
"PropertiesTopComponent.TechniqueLabel.text")); // NOI18N

    TechniqueTextField.setEditable(false);
    TechniqueTextField.setBackground(new java.awt.Color(255, 255, 255));

```

```

TechniqueTextField.setHorizontalAlignment(javax.swing.JTextField.LEFT);

TechniqueTextField.setText(org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class, "PropertiesTopComponent.TechniqueTextField.text")); // NOI18N
    TechniqueTextField.setMaximumSize(new java.awt.Dimension(196, 20));
    TechniqueTextField.setMinimumSize(new java.awt.Dimension(196, 20));
    TechniqueTextField.setPreferredSize(new java.awt.Dimension(196,
20));

    org.openide.awt.Mnemonics.setLocalizedText(IdentifierLabel,
org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class,
"PropertiesTopComponent.IdentifierLabel.text")); // NOI18N

    org.openide.awt.Mnemonics.setLocalizedText(NameLabel,
org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class,
"PropertiesTopComponent.NameLabel.text")); // NOI18N

    org.openide.awt.Mnemonics.setLocalizedText(SourceLabel,
org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class,
"PropertiesTopComponent.SourceLabel.text")); // NOI18N

    org.openide.awt.Mnemonics.setLocalizedText(QueryLabel,
org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class,
"PropertiesTopComponent.QueryLabel.text")); // NOI18N

    IdentifierTextField.setEditable(false);
    IdentifierTextField.setBackground(new java.awt.Color(255, 255,
255));

IdentifierTextField.setHorizontalAlignment(javax.swing.JTextField.LEFT);

IdentifierTextField.setText(org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class, "PropertiesTopComponent.IdentifierTextField.text")); // NOI18N
    IdentifierTextField.setMaximumSize(new java.awt.Dimension(196, 20));
    IdentifierTextField.setMinimumSize(new java.awt.Dimension(196, 20));
    IdentifierTextField.setPreferredSize(new java.awt.Dimension(196,
20));

    SourceTextField.setEditable(false);
    SourceTextField.setBackground(new java.awt.Color(255, 255, 255));
    SourceTextField.setHorizontalAlignment(javax.swing.JTextField.LEFT);

SourceTextField.setText(org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class, "PropertiesTopComponent.SourceTextField.text")); // NOI18N
    SourceTextField.setMaximumSize(new java.awt.Dimension(196, 20));
    SourceTextField.setMinimumSize(new java.awt.Dimension(196, 20));
    SourceTextField.setPreferredSize(new java.awt.Dimension(196, 20));

    QueryTextField.setEditable(false);
    QueryTextField.setBackground(new java.awt.Color(255, 255, 255));
    QueryTextField.setHorizontalAlignment(javax.swing.JTextField.LEFT);

QueryTextField.setText(org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class, "PropertiesTopComponent.QueryTextField.text")); // NOI18N
    QueryTextField.setMaximumSize(new java.awt.Dimension(196, 20));
    QueryTextField.setMinimumSize(new java.awt.Dimension(196, 20));
    QueryTextField.setPreferredSize(new java.awt.Dimension(196, 20));

    org.openide.awt.Mnemonics.setLocalizedText(DescriptionLabel,

```

```

org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class,
"PropertiesTopComponent.DescriptionLabel.text")); // NOI18N

DescriptionScrollPane.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneCo
nstants.HORIZONTAL_SCROLLBAR_NEVER);

        DescriptionTextArea.setEditable(false);
        DescriptionTextArea.setColumns(20);
        DescriptionTextArea.setFont(new java.awt.Font("Tahoma", 0, 11)); //
NOI18N
        DescriptionTextArea.setLineWrap(true);
        DescriptionTextArea.setRows(6);
        DescriptionTextArea.setWrapStyleWord(true);
        DescriptionScrollPane.setViewportViewView(DescriptionTextArea);

        NameTextField.setEditable(false);
        NameTextField.setBackground(new java.awt.Color(255, 255, 255));
        NameTextField.setHorizontalAlignment(javax.swing.JTextField.LEFT);

NameTextField.setText(org.openide.util.NbBundle.getMessage(PropertiesTopCom
ponent.class, "PropertiesTopComponent.NameTextField.text")); // NOI18N
        NameTextField.setMaximumSize(new java.awt.Dimension(196, 20));
        NameTextField.setMinimumSize(new java.awt.Dimension(196, 20));
        NameTextField.setPreferredSize(new java.awt.Dimension(196, 20));

        org.openide.awt.Mnemonics.setLocalizedText(CreatorLabel,
org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class,
"PropertiesTopComponent.CreatorLabel.text")); // NOI18N

        CreatorTextField.setEditable(false);
        CreatorTextField.setBackground(new java.awt.Color(255, 255, 255));

CreatorTextField.setHorizontalAlignment(javax.swing.JTextField.LEFT);

CreatorTextField.setText(org.openide.util.NbBundle.getMessage(PropertiesTop
Component.class, "PropertiesTopComponent.CreatorTextField.text_1")); //
NOI18N
        CreatorTextField.setMaximumSize(new java.awt.Dimension(196, 20));
        CreatorTextField.setMinimumSize(new java.awt.Dimension(196, 20));
        CreatorTextField.setPreferredSize(new java.awt.Dimension(196, 20));

        org.openide.awt.Mnemonics.setLocalizedText(DetailsLabel,
org.openide.util.NbBundle.getMessage(PropertiesTopComponent.class,
"PropertiesTopComponent.DetailsLabel.text")); // NOI18N

        DetailsTextField.setEditable(false);
        DetailsTextField.setBackground(new java.awt.Color(255, 255, 255));

DetailsTextField.setHorizontalAlignment(javax.swing.JTextField.LEFT);

DetailsTextField.setText(org.openide.util.NbBundle.getMessage(PropertiesTop
Component.class, "PropertiesTopComponent.DetailsTextField.text")); //
NOI18N
        DetailsTextField.setMaximumSize(new java.awt.Dimension(196, 20));
        DetailsTextField.setMinimumSize(new java.awt.Dimension(196, 20));
        DetailsTextField.setPreferredSize(new java.awt.Dimension(196, 20));

        javax.swing.GroupLayout PropertiesPanelLayout = new
javax.swing.GroupLayout(PropertiesPanel);
        PropertiesPanel.setLayout(PropertiesPanelLayout);

```

```

        PropertiesPanelLayout.setHorizontalGroup(

PropertiesPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
        .addGroup(PropertiesPanelLayout.createSequentialGroup())
            .addContainerGap()
            .addGroup(PropertiesPanelLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.LEADING)
                .addComponent(ImagePanel,
javax.swing.GroupLayout.PREFERRED_SIZE, 297,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(PropertiesPanelLayout.createSequentialGroup())
                        .addGroup(PropertiesPanelLayout.createParallelGroup
(javax.swing.GroupLayout.Alignment.LEADING, false)
                            .addComponent(IdentifierLabel,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                .addComponent(NameLabel,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                    .addComponent(SourceLabel,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                        .addComponent(QueryLabel,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                            .addComponent(CreationYearLabel,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                                .addComponent(DepictsLabel,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                                    .addComponent(ConsistsLabel,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                                        .addComponent(TypeLabel,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                                            .addComponent(TechniqueLabel,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                                                .addComponent(DescriptionLabel,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                                                    .addComponent(CommentaryLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 80,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                                                        .addComponent(CreatorLabel,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                                                                            .addComponent(DetailsLabel,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentP
lacement.RELATED)
                                .addGroup(PropertiesPanelLayout.createParallelGroup
(javax.swing.GroupLayout.Alignment.TRAILING, false)
                                    .addComponent(DescriptionScrollPane,
javax.swing.GroupLayout.Alignment.LEADING)
                                        .addComponent(TextField,
javax.swing.GroupLayout.Alignment.LEADING,

```

```

javax.swing.GroupLayout.DEFAULT_SIZE, 213, Short.MAX_VALUE)
    .addComponent(ConsistsTextField,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
    .addComponent(DepictsTextField,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
    .addComponent(QueryTextField,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
    .addComponent(SourceTextField,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
    .addComponent(TechniqueTextField,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
    .addComponent(CommentaryScrollPane)
    .addComponent(IdentifierTextField,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
    .addComponent(NameTextField,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
    .addComponent(CreatorTextField,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
    .addComponent(DetailsTextField,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
    .addComponent(CreationYearTextField,
javax.swing.GroupLayout.Alignment.LEADING)))
    .addContainerGap(14, Short.MAX_VALUE))
);
PropertiesPanelLayout.setVerticalGroup(

PropertiesPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
    .addGroup(PropertiesPanelLayout.createSequentialGroup())
        .addContainerGap()
        .addComponent(ImagePanel,
javax.swing.GroupLayout.PREFERRED_SIZE, 307,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
.RELATED)
        .addGroup(PropertiesPanelLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
            .addComponent(CreationYearLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(CreationYearTextField,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

```

```

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
.RELATED)
        .addGroup(PropertiesPanelLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
            .addComponent(IdentifierLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(IdentifierTextField,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
.RELATED)
        .addGroup(PropertiesPanelLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
            .addComponent(NameLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(NameTextField,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
.RELATED)
        .addGroup(PropertiesPanelLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
            .addComponent(SourceLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(SourceTextField,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
.RELATED)
        .addGroup(PropertiesPanelLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
            .addComponent(QueryLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(QueryTextField,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
.RELATED)
        .addGroup(PropertiesPanelLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
            .addComponent(DepictsLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(DepictsTextField,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
.RELATED)
        .addGroup(PropertiesPanelLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
            .addComponent(ConsistsLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(ConsistsTextField,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement

```

```

.RELATED)
        .addGroup(PropertiesPanelLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
            .addComponent(TypeLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(TextField,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
.RELATED)
        .addGroup(PropertiesPanelLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
            .addComponent(TechniqueLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(TechniqueTextField,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
.RELATED)
        .addGroup(PropertiesPanelLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
            .addComponent(CreatorLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(CreatorTextField,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
.RELATED)
        .addGroup(PropertiesPanelLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.BASELINE)
            .addComponent(DetailsLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(DetailsTextField,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
.UNRELATED)
        .addGroup(PropertiesPanelLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
PropertiesPanelLayout.createSequentialGroup()
                .addComponent(DescriptionScrollPane,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentP
lacement.UNRELATED))
            .addGroup(PropertiesPanelLayout.createSequentialGroup())
                .addComponent(DescriptionLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(78, 78, 78)))
        .addGroup(PropertiesPanelLayout.createParallelGroup(javax.s
wing.GroupLayout.Alignment.LEADING)
            .addComponent(CommentaryLabel,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE)

```



```

        .addComponent(CommentaryScrollPane,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addContainerGap()
    );

    PropertiesScrollPane.setViewportView(PropertiesPanel);

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
    this.setLayout(layout);
    layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(PropertiesScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
    );
    layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(PropertiesScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
    );
} // </editor-fold>

// Variables declaration - do not modify
private javax.swing.JLabel CommentaryLabel;
private javax.swing.JScrollPane CommentaryScrollPane;
private javax.swing.JTextArea CommentaryTextArea;
private javax.swing.JLabel ConsistsLabel;
private javax.swing.JTextField ConsistsTextField;
private javax.swing.JLabel CreationYearLabel;
private javax.swing.JTextField CreationYearTextField;
private javax.swing.JLabel CreatorLabel;
private javax.swing.JTextField CreatorTextField;
private javax.swing.JLabel DepictsLabel;
private javax.swing.JTextField DepictsTextField;
private javax.swing.JLabel DescriptionLabel;
private javax.swing.JScrollPane DescriptionScrollPane;
private javax.swing.JTextArea DescriptionTextArea;
private javax.swing.JLabel DetailsLabel;
private javax.swing.JTextField DetailsTextField;
private javax.swing.JLabel IdentifierLabel;
private javax.swing.JTextField IdentifierTextField;
private javax.swing.JPanel ImagePanel;
private javax.swing.JLabel NameLabel;
private javax.swing.JTextField NameTextField;
private javax.swing.JPanel PropertiesPanel;
private javax.swing.JScrollPane PropertiesScrollPane;
private javax.swing.JLabel QueryLabel;
private javax.swing.JTextField QueryTextField;
private javax.swing.JLabel SourceLabel;
private javax.swing.JTextField SourceTextField;
private javax.swing.JLabel TechniqueLabel;
private javax.swing.JTextField TechniqueTextField;
private javax.swing.JLabel TypeLabel;
private javax.swing.JTextField TypeTextField;
// End of variables declaration

```

```

private static final int IMAGE_SIZE = 280;

private Lookup.Result<ManMadeObject> objectResult = null;

/**
 * Called by the constructor.
 */
private void setup() {
    objectResult =
Utilities.actionsGlobalContext().lookupResult(ManMadeObject.class);
    objectResult.addLookupListener(this);
    ImagePanel.setLayout(new BorderLayout());
}

@Override
public void componentOpened() {}

@Override
public void componentClosed() {
    objectResult.removeLookupListener(this);
}

@Override
public void resultChanged(LookupEvent lookupEvent) {
    Collection<? extends ManMadeObject> allEntities =
objectResult.allInstances();
    if (allEntities.size() == 1) {

        ManMadeObject entity = allEntities.iterator().next();

        CreationYearTextField.setText(entity.getTimeSpan().toString());
        CreationYearTextField.setCaretPosition(0);

        IdentifierTextField.setText(entity.getIdentifier());
        IdentifierTextField.setCaretPosition(0);

        NameTextField.setText(entity.getName());
        NameTextField.setCaretPosition(0);

        SourceTextField.setText(entity.getSourceName());
        SourceTextField.setCaretPosition(0);

        QueryTextField.setText(entity.getQueryName());
        QueryTextField.setCaretPosition(0);

        DepictsTextField.setText(entity.getDepicts());
        DepictsTextField.setCaretPosition(0);

        ConsistsTextField.setText(entity.getConsists());
        ConsistsTextField.setCaretPosition(0);

        TypeTextField.setText(entity.getType());
        TypeTextField.setCaretPosition(0);

        TechniqueTextField.setText(entity.getTechnique());
        TechniqueTextField.setCaretPosition(0);

        CreatorTextField.setText(entity.getCreator());
        CreatorTextField.setCaretPosition(0);

        DetailsTextField.setText(entity.getObject());
    }
}

```

```

        DetailsTextField.setCaretPosition(0);

        DescriptionTextArea.setText(entity.getDescription());
        DescriptionTextArea.setCaretPosition(0);

        CommentaryTextArea.setText(entity.getCuratorial());
        CommentaryTextArea.setCaretPosition(0);

        ImagePanel.removeAll();
        BufferedImage thumb = entity.getThumb(IMAGE_SIZE);
        if (thumb != null) {
            JLabel image = new JLabel(new ImageIcon(thumb));
            ImagePanel.add(image, BorderLayout.CENTER);
        }
        ImagePanel.revalidate();
        ImagePanel.repaint();
    }
}

void writeProperties(java.util.Properties p) {
    p.setProperty("version", "1.0");
}

void readProperties(java.util.Properties p) {
    String version = p.getProperty("version");
}
}

```

12.5 Queries Module

This section contains all Java code from the Queries module.

12.5.1 QueryBuilder.java

```
package org.jghill.timelinevisualizerqueriesbuilder;

import org.jghill.timelinevisualizerqueries.QueryShell;
import org.jghill.timelinevisualizersources.Source;

/**
 * A singleton for building queries.
 *
 * @author JGHill
 */
public class QueryBuilder {

    private static final QueryBuilder BUILDER = new QueryBuilder();

    private QueryBuilder() {}

    public static QueryBuilder getInstance() {
        return BUILDER;
    }

    /**
     * A method for building queries from a QuerySettings object.
     *
     * @param settings the object containing the settings.
     * @return an object.
     */
    public QueryShell buildQuery(QuerySettings settings) {

        Source src = settings.source;
        QueryTranslator trans;

        switch(src.getSourceType()){
            case "SPARQL Endpoint":
                trans = new SPARQLTranslator();
                return trans.translate(settings);
            default:
                return null;
        }
    }
}
```

12.5.2 QuerySettings.java

```
package org.jghill.timelinesvisualizerqueriesbuilder;

import org.jghill.timelinevisualizersources.Source;

/**
 * An object for holding settings for queries.
 *
 * @author JGHill
 */
public class QuerySettings {

    public final Source source;
    public final String queryName;

    public final String creationStartDate;
    public final String creationEndDate;

    public final boolean hasNameCheck;
    public final String name;

    public final boolean hasIdentifierCheck;
    public final String identifier;

    public final boolean hasDepictionCheck;
    public final String depiction;

    public final boolean hasConsistsCheck;
    public final String consists;

    public final boolean hasTypeCheck;
    public final String type;

    public final boolean hasTechniqueCheck;
    public final String technique;

    public final boolean hasCreatorCheck;
    public final String creator;

    public final boolean hasLimitCheck;
    public final String limit;

    public final boolean hasImageCheck;

    public QuerySettings(

        Source source,
        String queryName,

        String creationStartDate,
        String creationEndDate,

        boolean hasNameCheck,
        String name,

        boolean hasIdentifierCheck,
        String identifier,

        boolean hasDepictionCheck,
```

```

        String depiction,

        boolean hasConsistsCheck,
        String consists,

        boolean hasTypeCheck,
        String type,

        boolean hasTechniqueCheck,
        String technique,

        boolean hasCreatorCheck,
        String creator,

        boolean hasLimitCheck,
        String limit,

        boolean hasImageCheck
    ) {
        this.source = source;
        this.queryName = queryName;

        this.creationStartDate = creationStartDate;
        this.creationEndDate = creationEndDate;

        this.hasNameCheck = hasNameCheck;
        this.name = name;

        this.hasIdentifierCheck = hasIdentifierCheck;
        this.identifier = identifier;

        this.hasDepictionCheck = hasDepictionCheck;
        this.depiction = depiction;

        this.hasConsistsCheck = hasConsistsCheck;
        this.consists = consists;

        this.hasTypeCheck = hasTypeCheck;
        this.type = type;

        this.hasTechniqueCheck = hasTechniqueCheck;
        this.technique = technique;

        this.hasCreatorCheck = hasCreatorCheck;
        this.creator = creator;

        this.hasLimitCheck = hasLimitCheck;
        this.limit = limit;

        this.hasImageCheck = hasImageCheck;
    }
}

```

12.5.3 SPARQLTranslator.java

```
package org.jghill.timelinesvisualizerqueriesbuilder;

import org.jghill.timelinevisualizerqueries.QueryShell;
import org.jghill.timelinevisualizerqueries.SPARQLQueryShell;
import org.jghill.timelinevisualizersources.SPARQLEndpoint;

/**
 * An implementation of QueryTranslator for SPARQL queries.
 *
 * @author JGHill
 */
public class SPARQLTranslator implements QueryTranslator {

    private QuerySettings settings;
    private SPARQLEndpoint sparql;

    private static final String BMO = "bmo:
<http://collection.britishmuseum.org/id/ontology/> \n";
    private static final String CRM = "crm: ";
    private static final String EDAN = "edan:
<http://edan.si.edu/saam/id/ontologies/> \n";
    private static final String RDF = "rdfs:
<http://www.w3.org/2000/01/rdf-schema#> \n";
    private static final String SKOS = "skos:
<http://www.w3.org/2004/02/skos/core#> \n";
    private static final String SKOS2 = "skos2:
<http://www.w3.org/2008/05/skos#> \n";
    private static final String THES = "thes:
<http://collection.britishmuseum.org/id/thesauri/> \n";
    private static final String XML = "xsd:
<http://www.w3.org/2001/XMLSchema#> \n";

    private static final String PREFIX = "PREFIX ";
    private static final String SELECT = "SELECT REDUCED ";
    private static final String WHERE = "WHERE { ";
    private static final String END = "} ";
    private static final String LIMIT = "LIMIT ";
    private static final String UNION = "UNION ";

    private static final String IDENTIFIER = "?identifier ";
    private static final String NAME = "?name ";
    private static final String OBJECT = "?object ";
    private static final String DEPICTION = "?depicts ";
    private static final String CONSISTS = "?consists ";
    private static final String TYPE = "?type ";
    private static final String TECHNIQUE = "?technique ";
    private static final String IMAGE = "?image ";
    private static final String DATE = "?date ";
    private static final String CREATOR = "?creator ";
    private static final String DESCRIPTION = "?description ";
    private static final String CURATORIAL = "?curatorial ";

    private static final String PRODUCTION = "?production ";
    private static final String PRODUCTION2 = "?production2 ";

    @Override
    public QueryShell translate(QuerySettings settings) {
        this.settings = settings;
    }
}
```

```

        sparql = (SPARQLEndpoint) settings.source;
        return new SPARQLQueryShell(
            build(),
            sparql.getWebAddress(),
            sparql.getCIDOCAddress(),
            settings.queryName,
            Integer.parseInt(settings.limit)
        );
    }

    /**
     * Builds the query from the components.
     *
     * @return the final query as a String.
     */
    private String build() {
        return
            prefix() + "\n\n" +
            select() + "\n\n" +
            WHERE + "\n\n" +
            whereClause() + "\n\n" +
            END + "\n\n" +
            limit();
    }

    /**
     * Builds the PREFIX part of the expression.
     */
    private String prefix() {
        return
            PREFIX + BMO +
            PREFIX + CRM + "<" + sparql.getCIDOCAddress() + "> \n" +
            PREFIX + EDAN +
            PREFIX + RDF +
            PREFIX + SKOS +
            PREFIX + SKOS2 +
            PREFIX + THES +
            PREFIX + XML;
    }

    /**
     * Builds the SELECT part of the expression.
     */
    private String select() {
        return
            SELECT + " \n" +
            IDENTIFIER + " \n" +
            CONSISTS + " \n" +
            DATE + " \n" +
            IMAGE + " \n" +
            OBJECT + " \n" +
            NAME + " \n" +
            TYPE + " \n" +
            DESCRIPTION + " \n" +
            CURATORIAL + " \n" +
            DEPICTION + " \n" +
            TECHNIQUE + " \n" +
            CREATOR + " \n";
    }
}

```



```

/**
 * Builds the where clause.
 */
private String whereClause() {
    String where = "";
    where += getDates() + "\n";
    where += getIdentifier() + "\n";
    where += getName() + "\n";
    where += getDepiction() + "\n";
    where += getConsists() + "\n";
    where += getType() + "\n";
    where += getTechnique() + "\n";
    where += getCreator() + "\n";
    where += getDescription() + "\n";
    where += getCuration() + "\n";
    where += getImage();
    return where;
}

/**
 * Returns the dates.
 *
 * @return the function for returning dates.
 */
private String getDates() {
    String dates = "";
    dates += PRODUCTION + " crm:P108_has_produced " + OBJECT + " .
\n";
    dates += "{ " + PRODUCTION +
"crm:P9_consists_of/crm:P4_has_time-span/rdfs:label " + DATE + " } \n";
    dates += UNION + "\n";
    dates += "{ " + PRODUCTION + " crm:P4_has_time-span/rdfs:label
" + DATE + " } .\n";

    if (!settings.creationStartDate.equals("")) {
        dates += "FILTER (xsd:integer(" + DATE + ") >= " +
            settings.creationStartDate + ") . \n";
    } else if (settings.creationStartDate.equals("") &&
        !settings.creationEndDate.equals("")) {
        dates += "FILTER (xsd:integer(" + DATE + ") > " + 0 + ") .
\n";
    }

    if (!settings.creationEndDate.equals("")) {
        dates += "FILTER (xsd:integer(" + DATE + ") <= " +
            settings.creationEndDate + ") . \n";
    }

    return dates;
}

/**
 * The identifier of the object.
 */
private String getIdentifier() {
    String triple = OBJECT +
"crm:P48_has_preferred_identifier/rdfs:label " + IDENTIFIER + " .\n";
    if (settings.hasIdentifierCheck) {
        triple += "FILTER (CONTAINS(LCASE(" + IDENTIFIER + "), \"\"
+ settings.identifier + "\")) . \n";
    }
}

```

```

        return triple;
    }

    /**
     * The name of the object.
     */
    private String getName() {
        String query = "";
        String triple = OBJECT + "crm:P102_has_title/rdfs:label " +
NAME;
        if (settings.hasNameCheck) {
            query += "{ " + triple + " }\n";
            query += "FILTER (CONTAINS(LCASE(" + NAME + "), \"\" +
settings.name + "\")) . \n";
        } else {
            query += triple + " . \n";
        }
        return query;
    }

    /**
     * The depiction of the object.
     */
    private String getDepiction() {
        String query = "";
        String triple = "";
        triple += "{ " + OBJECT + "crm:P62_depicts/skos:prefLabel " +
DEPICTION + " }\n";
        triple += UNION;
        triple += "{ " + OBJECT +
"crm:P128_carries/crm:P129_is_about/skos2:prefLabel " + DEPICTION + " }\n";
        if (settings.hasDepictionCheck) {
            query += triple;
            query += " . \n";
            query += "FILTER (CONTAINS(LCASE(" + DEPICTION + "), \"\" +
settings.depiction + "\")) . \n";
        } else {
            query += "OPTIONAL { " + triple + " } . \n";
        }
        return query;
    }

    /**
     * The material of the object.
     */
    private String getConsists() {
        String query = "";
        String triple = "";
        triple += "{ " + OBJECT + "crm:P45_consists_of/skos:prefLabel " +
+ CONSISTS + " }\n";
        triple += UNION;
        triple += "{ " + OBJECT + "edan:PE_medium_description " +
CONSISTS + " }\n";
        if (settings.hasConsistsCheck) {
            query += triple;
            query += "FILTER (CONTAINS(LCASE(" + CONSISTS + "), \"\" +
settings.consists + "\")) . \n";
        } else {
            query += triple + " . \n";
        }
        return query;
    }

```

```

    }

    /**
     * The type of the object.
     *
     * @return the type query string.
     */
    private String getType() {
        String query = "";
        String triple = "";
        triple += "{ " + OBJECT + "bmo:PX_object_type/skos:prefLabel "
+ TYPE + " }\n";
        triple += UNION + "\n";
        triple += "{ " + OBJECT +
"edan:PE_object_mainclass/skos2:prefLabel " + TYPE + " }\n";
        if (settings.hasTypeCheck) {
            query += triple;
            query += "FILTER (CONTAINS(LCASE(" + TYPE + "), \"\" +
settings.type + "\")) .\n";
        } else {
            query += triple + " .\n";
        }
        return query;
    }

    /**
     * The technique that created the object.
     *
     * @return the technique query part.
     */
    private String getTechnique() {
        String query = "";
        String triple = PRODUCTION +
"crm:P9_consists_of/crm:P32_used_general_technique/skos:prefLabel " +
TECHNIQUE + " ";
        if (settings.hasTechniqueCheck) {
            query += "{ " + triple + " }\n";
            query += "FILTER (CONTAINS(LCASE(" + TECHNIQUE + "), \"\" +
settings.technique + "\")) .\n";
        } else {
            query += "OPTIONAL { ";
            query += triple;
            query += " }\n";
        }
        return query;
    }

    /**
     * Returns the image that represents this object.
     *
     * @return the image URL.
     */
    private String getImage() {
        String query = "";
        String triple = OBJECT + "crm:P138i_has_representation " +
IMAGE;
        if (settings.hasImageCheck) {
            query += triple;
            query += " .\n";
        } else {
            query += "OPTIONAL { ";

```

```

        query += triple;
        query += " }\n";
    }
    return query;
}

/**
 * Returns the creator of this object.
 *
 * @return the creator name.
 */
private String getCreator() {
    String query = "";
    String triple = "OPTIONAL { ";
    triple += PRODUCTION2 + " crm:P108_has_produced " + OBJECT +
" .\n";
    triple += "{ " + PRODUCTION2 +
"crm:P14_carried_out_by/crm:P1_is_identified_by/rdfs:label " + CREATOR + " }
\n";
    triple += UNION + " \n";
    triple += "{ " + PRODUCTION2 +
"crm:P9_consists_of/crm:P14_carried_out_by/skos:prefLabel " + CREATOR +
" } } .\n";
    if (settings.hasCreatorCheck) {
        query += triple;
        query += "FILTER (CONTAINS(LCASE(" + CREATOR + "), \"" +
settings.creator + "\")) . \n";
    } else {
        query += triple;
    }
    return query;
}

/**
 * Line for obtaining an object description.
 *
 * @return the object description request.
 */
private String getDescription() {
    return "OPTIONAL { " + OBJECT + "bmo:PX_physical_description "
+ DESCRIPTION + " }\n";
}

/**
 * Line for obtaining a curators comments.
 *
 * @return the curatorial comment request.
 */
private String getCuration() {
    return "OPTIONAL { " + OBJECT + "bmo:PX_curatorial_comment " +
CURATORIAL + " }\n";
}

/**
 * Builds the LIMIT part of expression.
 *
 * @return the limit command.
 */
private String limit() {
    return LIMIT + (Integer.parseInt(settings.limit));
}

```

}

12.5.4 SPARQLQueryShell.java

```
package org.jghill.timelinevisualizerqueries;

import org.apache.jena.atlas.web.HttpException;
import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.jghill.timelinevisualizerentities.ManMadeObject;
import org.jghill.timelinevisualizerentitiescollection.EntitiesCollection;
import org.netbeans.api.io.IOProvider;
import org.netbeans.api.io.InputOutput;

/**
 * A concrete implementation of the QueryShell class representing a SPARQL
 * query.
 *
 * @author JGHill
 */
public class SPARQLQueryShell extends QueryShell {

    private final String queryString;
    private final String service;
    private final String cidoc;
    private final int limit;

    private static final String QUERY_TYPE = "SPARQL Endpoint";

    /**
     * The constructor.
     *
     * @param queryString holding the SPARQL query.
     * @param service with the URI for the service address.
     * @param cidoc implementation used.
     * @param name of the Source.
     * @param limit the number of records required.
     */
    public SPARQLQueryShell(
        String queryString,
        String service,
        String cidoc,
        String name,
        int limit
    ) {
        this.queryString = queryString;
        this.service = service;
        this.cidoc = cidoc;
        this.limit = limit;
        super.setQueryName(name);
    }

    @Override
    public EntitiesCollection run() {
        output(queryString);
        return getResults(QueryFactory.create(queryString));
    }
}
```

```

/**
 * Executes the query and returns the results.
 *
 * @param query the QueryFactory.
 * @return the entities.
 */
private EntitiesCollection getResults(Query query) throws HttpException
{
    output("connecting to endpoint");
    output("");

    try(QueryExecution qexec =
QueryExecutionFactory.sparqlService(service, query)) {
        ResultSet results = qexec.execSelect();
        return buildEntities(results, qexec);
    } catch (Exception ex) {
        return null;
    }

}

/**
 * Builds the collection of entities.
 *
 * @param results the ResultSet
 * @return the entities.
 */
private EntitiesCollection buildEntities(
    ResultSet results,
    QueryExecution qexec
) {
    output("getting results");
    output("");

    EntitiesCollection entities;
    entities = new EntitiesCollection(this.getQueryName());

    for(; results.hasNext();) {

        QuerySolution soln = results.next();

        String identity = "";
        if (soln.get("identifier") != null) {
            identity = soln.get("identifier").toString().trim();
        }

        String title = "";
        if (soln.get("name") != null) {
            title = soln.get("name").toString().trim();
        } else {
            title = identity;
        }

        String depicts = "";
        if (soln.get("depicts") != null) {
            depicts = soln.get("depicts").toString().trim();
        }

        String consists = "";

```

```

if (soln.get("consists") != null) {
    consists = soln.get("consists").toString().trim();
}

String type = "";
if (soln.get("type") != null) {
    type = soln.get("type").toString().trim();
}

String technique = "";
if (soln.get("technique") != null) {
    technique = soln.get("technique").toString().trim();
}

String image = "";
if (soln.get("image") != null) {
    image = soln.get("image").toString();
}

String year = "";
if (soln.get("date") != null) {
    year = soln.get("date").toString();
}

String creator = "";
if (soln.get("creator") != null) {
    creator = soln.get("creator").toString().trim();
}

String object = "";
if (soln.get("object") != null) {
    object = soln.get("object").toString();
}

String description = "";
if (soln.get("description") != null) {
    description = soln.get("description").toString().trim();
}

String curatorial = "";
if (soln.get("curatorial") != null) {
    curatorial = soln.get("curatorial").toString().trim();
}

ManMadeObject thing = new ManMadeObject(
    title,
    identity,
    service,
    super.getQueryName(),
    depicts,
    consists,
    type,
    technique,
    image,
    year,
    creator,
    object,
    description,
    curatorial
);

```



```

        if (!entities.getCollectionSet().contains(thing)) {
            entities.addThing(thing);

            output("Identifier   : " + identity.trim());
            output("Title       : " + title.trim());
            output("Depicts     : " + depicts.trim());
            output("Consists   : " + consists.trim());
            output("Type       : " + type.trim());
            output("Technique  : " + technique.trim());
            output("Image      : " + image.trim());
            output("Date       : " + year.trim());
            output("Creator    : " + creator.trim());
            output("Object     : " + object.trim());
            output("Description : " + description.trim());
            output("Curatorial : " + curatorial.trim());
            output("");
        }
    }

    qexec.abort();
    qexec.close();

    return entities;
}

/**
 * Returns the type of the query.
 *
 * @return The query type.
 */
@Override
public String getQueryType() {
    return QUERY_TYPE;
}

/**
 * Returns the query as a String.
 *
 * @return the query string.
 */
public String getQueryString() {
    return queryString;
}

/**
 * Returns the service address.
 *
 * @return the address.
 */
public String getServiceAddress() {
    return service;
}

/**
 * Returns the CIDOC implementation address.
 *
 * @return the CIDOC implementation.
 */

```

```

public String getCIDOCAddress() {
    return cidoc;
}

/**
 * Returns the limit passed to this query.
 *
 * @return the limit.
 */
public int getLimit() {
    return limit;
}

/**
 * Outputs an explanation of the action.
 *
 * @param text toString of the returned entity.
 */
private void output(String text) {
    InputOutput io = IOProvider.getDefault().getIO("Main", false);
    io.getOut().println(text);
}
}

```

12.5.5 QueriesCollection.java

```
package org.jghill.timelinevisualizerqueriescollection;

import java.util.Iterator;
import java.util.SortedSet;
import java.util.TreeSet;
import org.jghill.timelinevisualizerqueries.QueryShell;

/**
 * A collection for holding a set of AbstractQueries.
 *
 * @author JGHill
 */
public class QueriesCollection implements Iterable<QueryShell> {

    private final SortedSet<QueryShell> collection;

    public QueriesCollection() {
        collection = new TreeSet<>();
    }

    /**
     * A method for adding new queries into the collection.
     *
     * @param q A query.
     */
    public void addQuery(QueryShell q) {
        collection.add(q);
    }

    /**
     * Delete a query from the collection & return 'true' to confirm
     *
     * @param query The query to be deleted.
     * @return confirmation that a query has been added.
     */
    public boolean deleteQuery(QueryShell query) {
        return collection.remove(query);
    }

    /**
     * Return the entire collection.
     *
     * @return the query collection.
     */
    public SortedSet<QueryShell> getCollectionSet() {
        return collection;
    }

    /**
     * A method for returning a count of the queries in the collection.
     *
     * @return the count of queries.
     */
    public int getCount() {
        return collection.size();
    }

    @Override
```

```

public Iterator iterator() {
    return collection.iterator();
}

/**
 * Returns the collection as an array.
 *
 * @return the collection in array form.
 */
public QueryShell[] collectionToArray() {
    QueryShell[] queries = new QueryShell[collection.size()];
    queries = collection.toArray(queries);
    return queries;
}
}

```

12.6 Sources Module

This section contains all Java code from the Sources module.

12.6.1 SPARQLEndpoint.java

```
package org.jghill.timelinevisualizersources;

/**
 * A concrete class representing a SPARQL endpoint.
 *
 * @author JGHill
 */
public class SPARQLEndpoint extends Source {

    private static final String SOURCE_TYPE = "SPARQL Endpoint";
    private String webAddress;
    private String cidocAddress;

    /**
     * Constructor for the SPARQL endpoint.
     *
     * @param name A name for the source.
     * @param webAddress the web address.
     * @param cidoc the CIDOC CRM address.
     */
    public SPARQLEndpoint(
        String name,
        String webAddress,
        String cidoc
    ) {
        super.setSourceName(name);
        this.webAddress = webAddress;
        this.cidocAddress = cidoc;
    }

    /**
     * Returns the type of the source.
     *
     * @return The source type of this source.
     */
    @Override
    public String getSourceType() {
        return SOURCE_TYPE;
    }

    /**
     * @return The web address of the SPARQL endpoint.
     */
    public String getWebAddress() {
        return webAddress;
    }

    /**
     * @param address The web address.
     */
    public void setWebAddress(String address) {
```

```
        webAddress = address;
    }

    /**
     * @return the CIDOC CRM address.
     */
    public String getCIDOCAddress() {
        return cidocAddress;
    }

    /**
     * @param address The CIDOC CRM address.
     */
    public void setCIDOCAddress(String address) {
        cidocAddress = address;
    }
}
```

12.6.2 SourceCollection.java

```
package org.jghill.timelinevisualizersources;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.SortedSet;
import java.util.TreeSet;
import javax.swing.ComboBoxModel;
import javax.swing.DefaultComboBoxModel;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.xpath.XPathExpressionException;
import org.jghill.timelinevisualizersourcesxml.SourceManagerXMLParser;
import org.jghill.timelinevisualizersourcesxml.SourceManagerXMLParserImpl;
import org.jghill.timelinevisualizersourcesxml.SourceManagerXMLWriter;
import org.jghill.timelinevisualizersourcesxml.SourceManagerXMLWriterImpl;
import org.openide.util.Exceptions;
import org.openide.util.Lookup;
import org.openide.util.lookup.AbstractLookup;
import org.openide.util.lookup.InstanceContent;
import org.xml.sax.SAXException;

/**
 * A singleton pattern holding a collections of sources.
 *
 * @author JGHill
 */
public class SourceCollection implements Lookup.Provider {

    private static final SortedSet<Source> SOURCES = new TreeSet<>();
    private static final SourceCollection COLLECTION = new
SourceCollection();
    private static final InstanceContent IC = new InstanceContent();
    private static final Lookup LOOKUP = new AbstractLookup(IC);

    private static SourceManagerXMLWriter xmlWriter;

    public static boolean loaded = false;

    /**
     * Returns the single instance of this singleton pattern.
     *
     * @return this SourceCollection.
     */
    public static SourceCollection getInstance() {
        if (!loaded) {loadXML();}
        return COLLECTION;
    }

    /**
     * Return the entire collection.
     *
     * @return the source collection.
     */
    public static SortedSet<Source> getSourceCollectionSet() {
        return SOURCES;
    }
}
```

```

/**
 * Add a source to the collection.
 *
 * @param source The new source to be added.
 */
public static void addSource(Source source) {
    SOURCES.add(source);
    IC.add(source);
    saveXML();
}

/**
 * Delete a source from the collection & return 'true' to confirm.
 *
 * @param source The source to be deleted.
 */
public static void deleteSource(Source source) {
    SOURCES.remove(source);
    IC.remove(source);
    saveXML();
}

/**
 * Get a Source from a name.
 *
 * @param name of the Source.
 * @return the Source.
 */
public static Source getSource(String name) {
    Source source = null;
    for (Source s : SOURCES) {
        if (s.getSourceName().equalsIgnoreCase(name)) {
            source = s;
        }
    }
    return source;
}

/**
 * Returns the number of sources in the SourceCollection.
 *
 * @return the size of the collection.
 */
public static int getSize() {
    return SOURCES.size();
}

/**
 * Returns the collection as an array.
 *
 * @return the collection in array form.
 */
public static Source[] collectionToArray() {
    Source[] sources = new Source[SOURCES.size()];
    sources = SOURCES.toArray(sources);
    return sources;
}

/**
 * Returns a ComboBox based on the existing Sources.
 *

```



```

    * @return the ComboBox.
    */
    public static ComboBoxModel getSourceComboBoxModel() {
        if (!loaded) {loadXML();}
        List<String> sources = new ArrayList<>();
        sources.add("Select . . .");
        for (Source source : SOURCES.toArray(new Source[0])) {
            sources.add(source.getSourceName());
        }
        return new DefaultComboBoxModel(sources.toArray(new String[0]));
    }

    @Override
    public Lookup getLookup() {
        return LOOKUP;
    }

    /**
     * Saves the Source Collection as an XML document.
     */
    private static void saveXML() {
        try {
            xmlWriter = new SourceManagerXMLWriterImpl();
            xmlWriter.build();
            xmlWriter.print();
        } catch (ParserConfigurationException ex) {}
    }

    /**
     * Loads the Sources from an XML file.
     */
    private static void loadXML() {
        File f = new File("Data/Source Manager/Source Manager.xml");
        if (f.exists()) {
            try {
                SourceManagerXMLParser parser;
                parser = new SourceManagerXMLParserImpl(f);
                List<Source> sources = parser.parseSources();
                sources.stream().forEach(SourceCollection::addSource);
                loaded = true;
            } catch (ParserConfigurationException | SAXException |
IOException | XPathExpressionException ex) {
                Exceptions.printStackTrace(ex);
            }
        }
    }
}

```

12.6.3 SourceManagementTool.java

```
package org.jghill.timelinevisualizersourcesgui;

import org.jghill.timelinevisualizersources.Source;
import org.jghill.timelinevisualizersources.SourceCollection;
import org.jghill.timelinevisualizersources.SourceTableModel;

/**
 * A dialog box for managing sources.
 *
 * @author JGHill
 */
public class SourceManagementTool extends javax.swing.JDialog {

    /**
     * Creates new form SourceManagementTool
     */
    public SourceManagementTool(
        java.awt.Frame parent,
        boolean modal
    ) {
        super(parent, modal);
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the
     form.
     * WARNING: Do NOT modify this code. The content of this method is
     always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        SourceManagementNewButton = new javax.swing.JButton();
        SourceManagementEditButton = new javax.swing.JButton();
        SourceManagementDeleteButton = new javax.swing.JButton();
        SourceManagementScrollPane = new javax.swing.JScrollPane();
        SourceManagementSourceTable = new javax.swing.JTable();
        SourceManagementCloseButton = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);

        setTitle(org.openide.util.NbBundle.getMessage(SourceManagementTool.class,
"SourceManagementTool.title")); // NOI18N
        setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        setIconImages(null);
        setLocationByPlatform(true);
        setMaximumSize(new java.awt.Dimension(400, 200));
        setMinimumSize(new java.awt.Dimension(400, 200));
        setName("Source Management Tool"); // NOI18N
        setPreferredSize(new java.awt.Dimension(400, 200));
        setResizable(false);

        org.openide.awt.Mnemonics.setLocalizedText(SourceManagementNewButton,
```

```

org.openide.util.NbBundle.getMessage(SourceManagementTool.class,
"SourceManagementTool.SourceManagementNewButton.text")); // NOI18N

SourceManagementNewButton.setToolTipText(org.openide.util.NbBundle.getMessa
ge(SourceManagementTool.class,
"SourceManagementTool.SourceManagementNewButton.toolTipText")); // NOI18N
SourceManagementNewButton.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        SourceManagementNewButtonActionPerformed(evt);
    }
});

org.openide.awt.Mnemonics.setLocalizedText(SourceManagementEditButton,
org.openide.util.NbBundle.getMessage(SourceManagementTool.class,
"SourceManagementTool.SourceManagementEditButton.text")); // NOI18N
SourceManagementEditButton.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        SourceManagementEditButtonActionPerformed(evt);
    }
});

org.openide.awt.Mnemonics.setLocalizedText(SourceManagementDeleteButton,
org.openide.util.NbBundle.getMessage(SourceManagementTool.class,
"SourceManagementTool.SourceManagementDeleteButton.text")); // NOI18N
SourceManagementDeleteButton.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        SourceManagementDeleteButtonActionPerformed(evt);
    }
});

SourceManagementScrollPane.setHorizontalScrollBarPolicy(javax.swing.ScrollP
aneConstants.HORIZONTAL_SCROLLBAR_NEVER);

SourceManagementScrollPane.setVerticalScrollBarPolicy(javax.swing.ScrollPan
eConstants.VERTICAL_SCROLLBAR_ALWAYS);
SourceManagementScrollPane.setCursor(new
java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
SourceManagementScrollPane.setHorizontalScrollBar(null);
SourceManagementScrollPane.setMaximumSize(new
java.awt.Dimension(380, 140));
SourceManagementScrollPane.setMinimumSize(new
java.awt.Dimension(380, 140));
SourceManagementScrollPane.setPreferredSize(new
java.awt.Dimension(380, 140));

SourceManagementSourceTable.setModel(new
SourceTableModel(SourceCollection.getInstance()));

SourceManagementSourceTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZ
E_LAST_COLUMN);
SourceManagementSourceTable.setMaximumSize(new
java.awt.Dimension(380, 140));
SourceManagementSourceTable.setMinimumSize(new
java.awt.Dimension(380, 140));
SourceManagementSourceTable.setPreferredSize(new

```

```

java.awt.Dimension(380, 140));

SourceManagementScrollPane.setViewportView(SourceManagementSourceTable);

org.openide.awt.Mnemonics.setLocalizedText(SourceManagementCloseButton,
org.openide.util.NbBundle.getMessage(SourceManagementTool.class,
"SourceManagementTool.SourceManagementCloseButton.text")); // NOI18N
    SourceManagementCloseButton.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        SourceManagementCloseButtonActionPerformed(evt);
    }
});

    javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(18, 18, 18)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(SourceManagementScrollPane,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addGroup(layout.createSequentialGroup()
                    .addComponent(SourceManagementNewButton)
                    .addGap(18, 18, 18)
                    .addComponent(SourceManagementEditButton)
                    .addGap(18, 18, 18)
                    .addComponent(SourceManagementDeleteButton)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(SourceManagementCloseButton,
javax.swing.GroupLayout.PREFERRED_SIZE, 59,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGap(18, 18, 18)
                )
            )
        );
    layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(18, 18, 18)
            .addComponent(SourceManagementScrollPane,
javax.swing.GroupLayout.PREFERRED_SIZE, 140,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(SourceManagementNewButton)
                .addComponent(SourceManagementEditButton)
                .addComponent(SourceManagementDeleteButton)
                .addComponent(SourceManagementCloseButton)
            )
            .addGap(18, 18, 18)
        )
    );

```

```

        setSize(new java.awt.Dimension(416, 235));
        setLocationRelativeTo(null);
    } // </editor-fold>

    private void
    SourceManagementNewButtonActionPerformed(java.awt.event.ActionEvent evt) {
        SourceManagementNew s;
        s = new SourceManagementNew(new javax.swing.JFrame(), true);
        s.setSourceTableModel((SourceTableModel)
    SourceManagementSourceTable.getModel());
        s.setVisible(true);
    }

    private void
    SourceManagementDeleteButtonActionPerformed(java.awt.event.ActionEvent evt)
    {
        int row = SourceManagementSourceTable.getSelectedRow();
        if(row != -1) {
            SourceTableModel s =
    (SourceTableModel)SourceManagementSourceTable.getModel();
            s.deleteSource(row);
        }
    }

    private void
    SourceManagementCloseButtonActionPerformed(java.awt.event.ActionEvent evt)
    {
        this.dispose();
    }

    private void
    SourceManagementEditButtonActionPerformed(java.awt.event.ActionEvent evt) {
        int row = SourceManagementSourceTable.getSelectedRow();
        if(row != -1) {
            SourceTableModel sourceTable;
            sourceTable =
    (SourceTableModel)SourceManagementSourceTable.getModel();
            Source src;
            src = sourceTable.returnSource(row);
            SourceManagementEdit editor;
            editor = new SourceManagementEdit(new javax.swing.JFrame(),
true, src);
            editor.setVisible(true);
        }
    }

    // Variables declaration - do not modify
    private javax.swing.JButton SourceManagementCloseButton;
    private javax.swing.JButton SourceManagementDeleteButton;
    private javax.swing.JButton SourceManagementEditButton;
    private javax.swing.JButton SourceManagementNewButton;
    private javax.swing.JScrollPane SourceManagementScrollPane;
    private javax.swing.JTable SourceManagementSourceTable;
    // End of variables declaration
}

```

12.6.4 SourceManagerXMLParserImpl.java

```
package org.jghill.timelinevisualizersourcesxml;

import java.io.File;
import java.io.IOException;
import java.util.LinkedList;
import java.util.List;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;
import org.jghill.timelinevisualizersources.SPARQLEndpoint;
import org.jghill.timelinevisualizersources.Source;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;

/**
 * An implementation of the Source Manager XML parser.
 *
 * @author JGHill
 */
public class SourceManagerXMLParserImpl implements SourceManagerXMLParser {

    private final Document doc;
    private final DocumentBuilder builder;
    private final XPath path;

    /**
     * The Constructor.
     *
     * @param f the file name.
     * @throws ParserConfigurationException
     * @throws SAXException
     * @throws IOException
     */
    public SourceManagerXMLParserImpl(File f) throws
    ParserConfigurationException, SAXException, IOException {
        DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
        builder = factory.newDocumentBuilder();
        doc = builder.parse(f);
        XPathFactory xFactory = XPathFactory.newInstance();
        path = xFactory.newXPath();
    }

    @Override
    public List<Source> parseSources() throws XPathExpressionException {
        List<Source> sources = new LinkedList<>();
        int sourcesCount;
        sourcesCount = Integer.parseInt(
            path.evaluate("count(/manager/source)", doc)
        );
        for (int i = 1; i <= sourcesCount; i++) {
            String type = path.evaluate("/manager/source[" + i + "]/type",
doc);
            if (type.equalsIgnoreCase("SPARQL Endpoint")) {
                String name = path.evaluate("/manager/source[" + i +

```

```

    "/name", doc);
        String uri = path.evaluate("/manager/source[" + i + "]/uri",
doc);
        String cidoc = path.evaluate("/manager/source[" + i +
"/cidoc", doc);
        Source s = new SPARQLEndpoint(
            name,
            uri,
            cidoc
        );
        sources.add(s);
    }

    }
    return sources;
}
}

```

12.6.5 SourceManagerXMLWriterImpl.java

```
package org.jghill.timelinevisualizersourcesxml;

import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.jghill.timelinevisualizersources.SPARQLEndpoint;
import org.jghill.timelinevisualizersources.Source;
import org.jghill.timelinevisualizersources.SourceCollection;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Text;

/**
 * An implementation of the Source Manager XML writer.
 *
 * @author JGHill
 */
public class SourceManagerXMLWriterImpl implements SourceManagerXMLWriter {

    final private DocumentBuilder builder;
    private Document doc;

    /**
     * Constructor.
     *
     * @throws ParserConfigurationException
     */
    public SourceManagerXMLWriterImpl() throws ParserConfigurationException {
        builder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
    }

    @Override
    public void build() {
        doc = builder.newDocument();
        Element rootElement = doc.createElement("manager");
        doc.appendChild(rootElement);
        createSourceList(rootElement);
    }

    @Override
    public void print() {
        Transformer trans;
        try {
            trans = TransformerFactory.newInstance().newTransformer();
            trans.setOutputProperty(OutputKeys.INDENT, "yes");
            trans.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "2");
            DOMSource source = new DOMSource(doc);
            StreamResult file = new StreamResult(new File("Data/Source Manager/Source Manager.xml"));
        }
    }
}
```



```

        trans.transform(source, file);
    } catch (TransformerException ex) {}
}

/**
 * Loops through the list of sources assigning xml to the root element.
 *
 * @param rootElement to assign sources to.
 */
private void createSourceList(Element rootElement) {
    Source[] sources = SourceCollection.collectionToArray();
    for (Source source : sources) {
        rootElement.appendChild(createSource(source));
    }
}

/**
 * Creates an xml implementation of the source.
 *
 * @param source to be transformed to xml.
 * @return the xml version of the source.
 */
private Element createSource(Source source) {
    Element e = doc.createElement("source");
    e.appendChild(createTextElement("name", "" +
source.getSourceName()));
    e.appendChild(createTextElement("type", "" +
source.getSourceType()));
    if (source instanceof SPARQLEndpoint) {
        e.appendChild(createTextElement("uri", "" + ((SPARQLEndpoint)
source).getWebAddress()));
        e.appendChild(createTextElement("cidoc", "" + ((SPARQLEndpoint)
source).getCIDOCAddress()));
    }
    return e;
}

/**
 * Creates a textual Element.
 *
 * @param name for the element.
 * @param text to be contained with the element.
 * @return the element.
 */
private Element createTextElement(
    String name,
    String text
) {
    Text t = doc.createTextNode(text);
    Element e = doc.createElement(name);
    e.appendChild(t);
    return e;
}
}

```

12.7 Viewer Module

This section contains all Java code from the Viewer module.

12.7.1 ViewerTopComponent.java

```
package org.jghill.timelinevisualizerviewer;

import java.io.File;
import java.io.IOException;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.xpath.XPathExpressionException;
import org.jghill.timelinesvisualizercollections.Collection;
import org.jghill.timelinesvisualizercollections.container.CollectionContainer;
import org.jghill.timelinesvisualizercollectionxml.CollectionXMLParser;
import org.jghill.timelinesvisualizercollectionxml.CollectionXMLParserImpl;
import org.netbeans.api.settings.ConvertAsProperties;
import org.openide.awt.ActionID;
import org.openide.explorer.ExplorerManager;
import org.openide.explorer.ExplorerUtils;
import org.openide.explorer.view.BeanTreeView;
import org.openide.nodes.AbstractNode;
import org.openide.windows.TopComponent;
import org.openide.util.NbBundle.Messages;
import org.xml.sax.SAXException;

/**
 * Viewer for the manipulation of Collections.
 */
@ConvertAsProperties(
    dtd = "-//org.jghill.timelinesvisualizercollectionsgui//CollectionView//EN",
    autostore = false
)
@TopComponent.Description(
    preferredID = "ViewerTopComponent",
    persistenceType = TopComponent.PERSISTENCE_ALWAYS
)
@TopComponent.Registration(mode = "viewer", openAtStartup = true)
@ActionID(category = "Window", id =
"org.jghill.timelinesvisualizercollectionsgui.ViewerTopComponent")
//@ActionReference(path = "Menu/Window" /*, position = 333 */)
@TopComponent.OpenActionRegistration(
    displayName = "#CTL_CollectionViewerAction",
    preferredID = "ViewerTopComponent"
)
@Messages({
    "CTL_CollectionViewerAction=Viewer",
    "CTL_CollectionViewerTopComponent=Viewer",
    "HINT_CollectionViewerTopComponent=This is a Viewer"
})
public final class ViewerTopComponent extends TopComponent implements
ExplorerManager.Provider {

    private final ExplorerManager manager = new ExplorerManager();
```

```

/**
 * Constructor.
 */
public ViewerTopComponent() {

    initComponents();
    setName(Bundle.CTL_CollectionViewerTopComponent());
    setToolTipText(Bundle.HINT_CollectionViewerTopComponent());

    putClientProperty(TopComponent.PROP_CLOSING_DISABLED, Boolean.TRUE);
    putClientProperty(TopComponent.PROP_DRAGGING_DISABLED,
Boolean.TRUE);
    putClientProperty(TopComponent.PROP_MAXIMIZATION_DISABLED,
Boolean.TRUE);
    putClientProperty(TopComponent.PROP_UNDOCKING_DISABLED,
Boolean.TRUE);

    setup();

}

/**
 * This method is called from within the constructor to initialize the
form.
 * WARNING: Do NOT modify this code. The content of this method is
always
 * regenerated by the Form Editor.
 */
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    ViewerScrollPane = new BeanTreeView();

    setToolTipText(org.openide.util.NbBundle.getMessage(ViewerTopComponent.clas
s, "ViewerTopComponent.toolTipText")); // NOI18N
    setMinimumSize(new java.awt.Dimension(250, 100));
    setOpaque(true);
    setPreferredSize(new java.awt.Dimension(250, 500));

    ViewerScrollPane.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstan
ts.HORIZONTAL_SCROLLBAR_NEVER);

    ViewerScrollPane.setToolTipText(org.openide.util.NbBundle.getMessage(Viewer
TopComponent.class, "ViewerTopComponent.ViewerScrollPane.toolTipText")); //
NOI18N
    ViewerScrollPane.setCursor(new
java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
    ViewerScrollPane.setHorizontalScrollBar(null);
    ViewerScrollPane.setMinimumSize(new java.awt.Dimension(250, 100));
    ViewerScrollPane.setPreferredSize(new java.awt.Dimension(250, 500));

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
    this.setLayout(layout);
    layout.setHorizontalGroup(

        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(ViewerScrollPane,
                javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                Short.MAX_VALUE)
    );
}

```

```

        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(ViewerScrollPane,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
        );
    }// </editor-fold>

    // Variables declaration - do not modify
    private javax.swing.JScrollPane ViewerScrollPane;
    // End of variables declaration

    /**
     * Called by the constructor.
     */
    private void setup() {
        associateLookup(ExplorerUtils.createLookup(manager,
getActionMap()));
        manager.setRootContext(new
AbstractNode(CollectionContainer.getChildren()));
        manager.getRootContext().setDisplayName("Collections");
        loadFromFile();
    }

    @Override
    public void componentOpened() {}

    @Override
    public void componentClosed() {}

    @Override
    protected void componentActivated() {
        ExplorerUtils.activateActions(manager, true);
    }

    @Override
    protected void componentDeactivated() {
        ExplorerUtils.activateActions(manager, false);
    }

    void writeProperties(java.util.Properties p) {
        p.setProperty("version", "1.0");
    }

    void readProperties(java.util.Properties p) {
        String version = p.getProperty("version");
    }

    @Override
    public ExplorerManager getExplorerManager() {
        return manager;
    }

    /**
     * Loads any .xml files from the Collections folder.
     */
    private void loadFromFile() {

```

```

File folder = new File("Data/Collections/");
File[] listOfFiles = folder.listFiles();

for (File file : listOfFiles) {
    if (file.getName().endsWith(".xml")) {
        try {
            CollectionXMLParser parser;
            parser = new CollectionXMLParserImpl(file);
            Collection collection = parser.parseCollection();
            CollectionContainer.addCollection(collection);
            CollectionContainer.addToLookup(collection);
        } catch (ParserConfigurationException | SAXException |
IOException | XPathExpressionException ex) {
            System.out.println("Error: " + ex);
        }
    }
}
}
}

```