

The University of Akron IdeaExchange@UAkron

Honors Research Projects

The Dr. Gary B. and Pamela S. Williams Honors
College

Summer 2016


srcMX: A GUI Application for srcML

Brian Kovacs

The University of Akron, bck25@zips.uakron.edu

Please take a moment to share how this work helps you [through this survey](#). Your feedback will be important as we plan further development of our repository.

Follow this and additional works at: http://ideaexchange.uakron.edu/honors_research_projects

 Part of the [Graphics and Human Computer Interfaces Commons](#), [Programming Languages and Compilers Commons](#), and the [Software Engineering Commons](#)

Recommended Citation

Kovacs, Brian, "srcMX: A GUI Application for srcML" (2016). *Honors Research Projects*. 395.
http://ideaexchange.uakron.edu/honors_research_projects/395

This Honors Research Project is brought to you for free and open access by The Dr. Gary B. and Pamela S. Williams Honors College at IdeaExchange@UAkron, the institutional repository of The University of Akron in Akron, Ohio, USA. It has been accepted for inclusion in Honors Research Projects by an authorized administrator of IdeaExchange@UAkron. For more information, please contact mjon@uakron.edu, uapress@uakron.edu.

srcMX: A GUI Application for srcML

Brian Kovacs

Department of Computer Science

Honors Research Project

Submitted to

The Honors College

Approved:

_____ Date _____
Honors Project Sponsor (signed)

Honors Project Sponsor (printed)

_____ Date _____
Reader (signed)

Reader (printed)

_____ Date _____
Reader (signed)

Reader (printed)

Accepted:

_____ Date _____
Department Head (signed)

Department Head (printed)

_____ Date _____
Honors Faculty Advisor (signed)

Honors Faculty Advisor (printed)

_____ Date _____
Dean, Honors College

Introduction:

For my Honors Research Project, I developed a GUI application in C++ using the Qt and Qt Quick frameworks. My application is called srcMX, and it utilizes the srcML command-line tool to convert and display source code using the srcML format. My goal is for srcMX to promote the manipulation and exploration of source code using srcML. I also hope that the user-friendly nature inherent to GUI applications allows srcMX to introduce a larger audience to the many features offered by srcML.

The srcML format is a representation for source code where elements of the abstract syntax for the language are identified by XML markup tags. The srcML command-line tool efficiently converts source code files to and from the srcML format with a lossless approach that preserves all the original code, formatting, and comments. Source code files can be parsed and translated at speeds of approximately twenty-five thousand lines per second. Languages currently supported by the parser include C, C++, C#, and Java [1].

The development of the srcML technology is led by Principal Investigators Dr. Michael L. Collard and Dr. Jonathan I. Maletic. In addition, the project is partially supported by a grant from the National Science Foundation (CNS 1305217) [2]. Additional information regarding srcML is available at the official website, www.srcml.org. An example of a simple C++ program converted to the srcML format is illustrated in figures 1.1 and 1.2.

Figure 1.1 Simple C++ program, rotate.cpp [3]

```
#include "rotate.h"

// rotate three values
void rotate(int& n1, int& n2, int& n3)
{
    // copy original values
    int tn1 = n1, tn2 = n2, tn3 = n3;

    // move
    n1 = tn3;
    n2 = tn1;
    n3 = tn2;
}
```

Figure 1.2 Corresponding srcML file, rotate.xml [3]

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<unit xmlns="http://www.srcML.org/srcML/src" xmlns:cpp="http://
www.srcML.org/srcML/cpp" revision="0.9.5" language="C++"
filename="rotate.cpp">
  <cpp:include>#<cpp:directive>include</cpp:directive>
<cpp:file>"rotate.h"</cpp:file></cpp:include>

  <comment type="line">// rotate three values</comment>
  <function><type><name>void</name></type> <name>rotate</
name><parameter_list>(<parameter><decl><type><name>int</
name><modifier>&</modifier></type> <name>n1</name></decl></
parameter>, <parameter><decl><type><name>int</name><modifier>&</
modifier></type> <name>n2</name></decl></parameter>,
<parameter><decl><type><name>int</name><modifier>&</modifier></type>
<name>n3</name></decl></parameter>)</parameter_list>
  <block>{
    <comment type="line">// copy original values</comment>
    <decl_stmt><decl><type><name>int</name></type> <name>tn1</
name> <init>= <expr><name>n1</name></expr></init></decl>, <decl><type
ref="prev"/><name>tn2</name> <init>= <expr><name>n2</name></expr></
init></decl>, <decl><type ref="prev"/><name>tn3</name> <init>=
<expr><name>n3</name></expr></init></decl>;</decl_stmt>

    <comment type="line">// move</comment>
    <expr_stmt><expr><name>n1</name> <operator>=</operator>
<name>tn3</name></expr>;</expr_stmt>
    <expr_stmt><expr><name>n2</name> <operator>=</operator>
<name>tn1</name></expr>;</expr_stmt>
    <expr_stmt><expr><name>n3</name> <operator>=</operator>
<name>tn2</name></expr>;</expr_stmt>
  }</block></function>
</unit>
```

A code base consisting of many source code files can be parsed and translated into a srcML archive. A srcML archive is a single XML file where the srcML for individual source code files is delineated by opening and closing unit tags. These unit tags encapsulate the srcML of every source code file, and they contain the file's name and language as shown in the example above. As a result, individual files within the srcML archive can be accessed sequentially by the file's unit number [1].

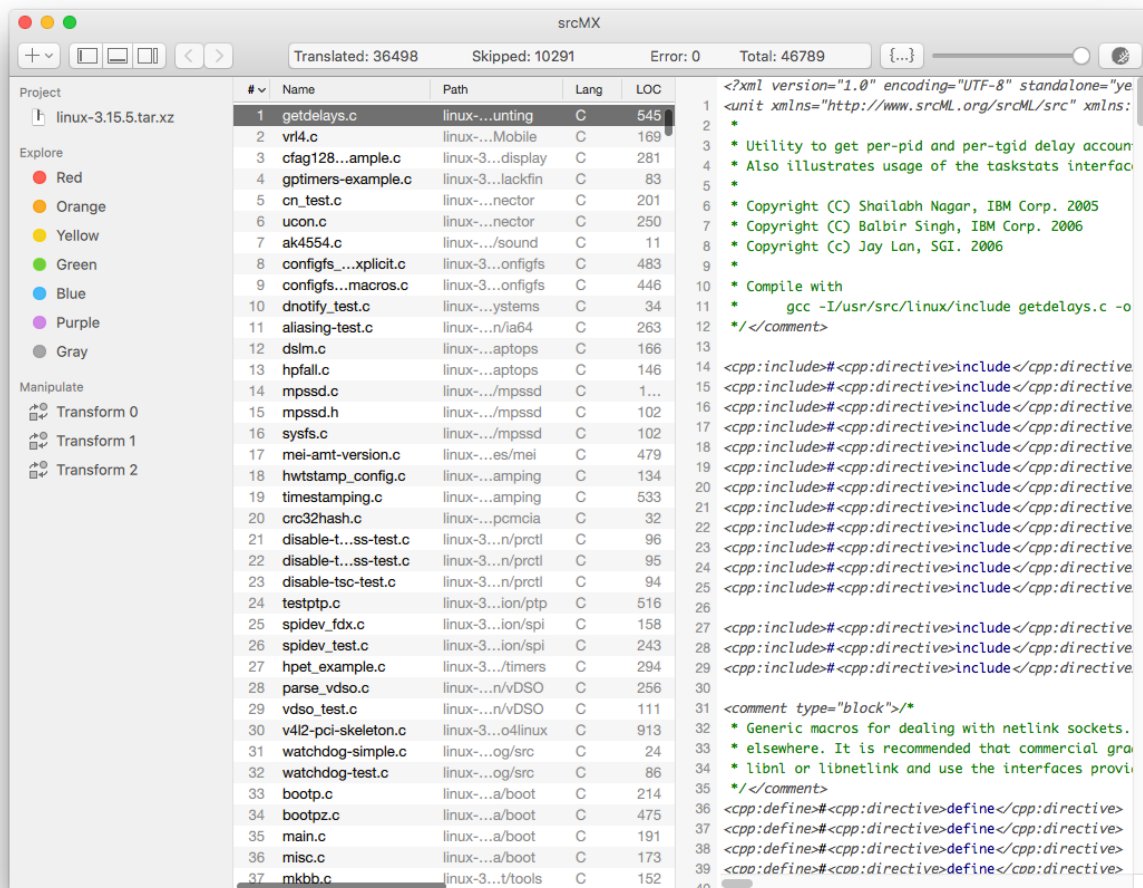
Once a source code file is converted to the srcML format, the XML structure of srcML can be utilized to perform different visualizations and transformations of the source code. A multitude of technologies such as XPath, XQuery, RelaxNG, XSchema, XSLT, DOM, and SAX can perform such activities as fact extraction, validation, and transformation [1]. While the srcML command-line tool is exceedingly efficient at translating source code files to and from the srcML format, users may need to utilize external tools and applications to explore and manipulate the results. In response, the srcMX application is the first that enables users to convert source code into the srcML format while also providing an interactive method for dynamically visualizing the results. In addition, my application also provides users with the ability to construct and perform queries on source code in the srcML format. These query results can be visualized in srcMX using both the srcML format and native source code.

srcMX Application:

The srcMX application presents the user with an interface that consists of several modular views. These views support a range of functionality, including such things as

project management and configuration, query configuration and visualization, and source-code visualization. In addition, the application also provides the user with information regarding the parsing and translation of files with srcml. Figure 2.1 provides an overview of the srcMX application after having parsed and translated the entire linux kernel.

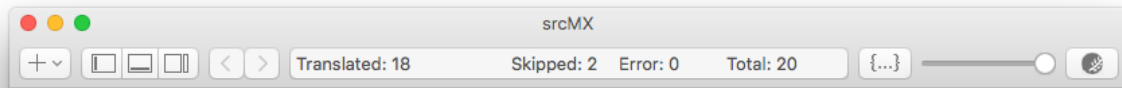
Figure 2.1 The Linux kernel open in the srcMX application



At the top of the application, the main toolbar contains a variety of controls. Here, the user can add files to the project, toggle the visibility of application views, and browse through the history of viewed files. To the right, a slider controls the visibility of srcML

tags in the code viewport. As shown in figure 2.2, the main toolbar also contains a large status bar that displays information about the project.

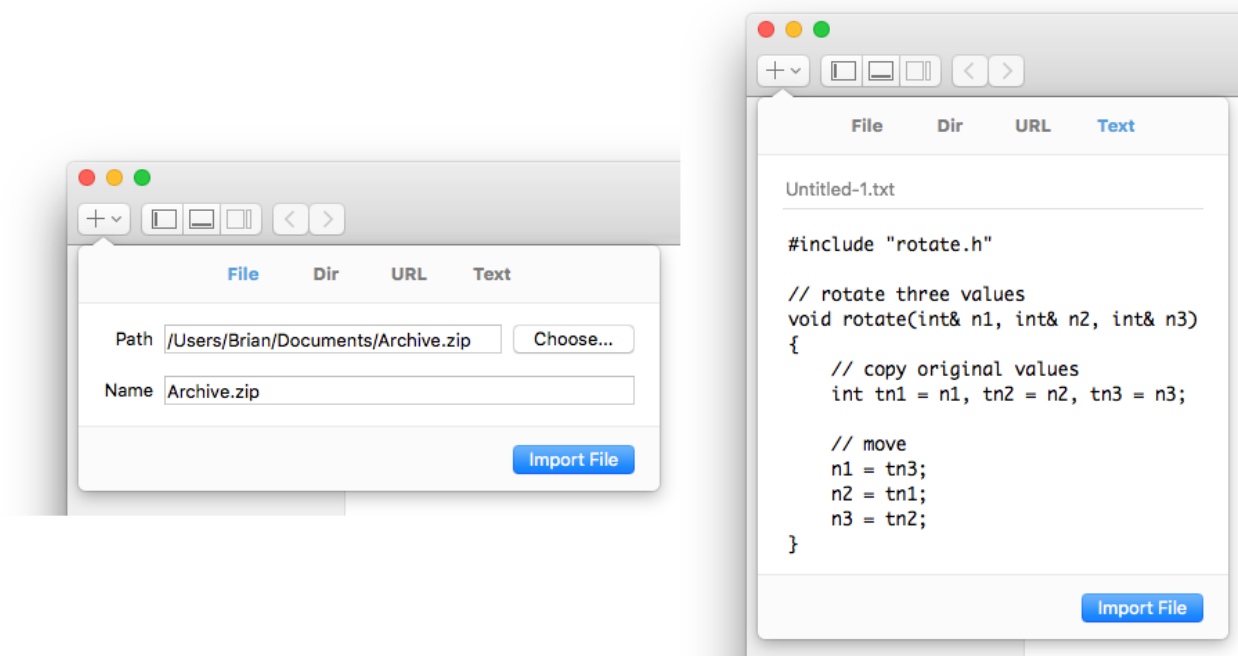
Figure 2.2 The main toolbar displays project information



During parsing, the status bar updates to display the name, lines of code, language, and hash of the most recently parsed file. The user can also stop the import process by clicking the corresponding button to the right of the bar. After file parsing has completed, the status bar displays the number of files translated, the number of files skipped, the number of errors encountered, and the total number of files parsed. As the user browses files within the project, the status bar displays the name of the currently selected file.

Users can add new files to a project by expanding the drop-down button in the main tool bar. This opens a window that allows the user to specify the type of file he or she wishes to add. Like the srcML command line tool, srcMX can accept individual files, directories containing multiple files, compressed archives of multiple files, and can even load files from a URL. Users can also add raw text to a project within the new file window.

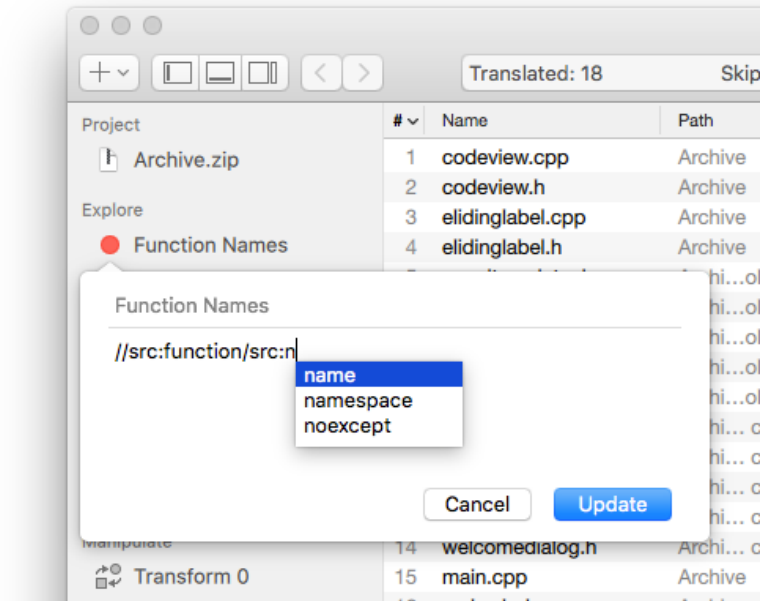
Figure 2.3 Adding source code files to the srcMX application



To the left of the application, a collapsible side bar is divided into three sections. Here, the user can view the list of input sources, queries, and transformations within the current project.

The srcMX application supports up to seven simultaneous queries within a project, each having a unique color tag and label for identification. Unlabeled queries take the name of their color tag. Queries can also be reordered via a click and drag mechanism. Reordering a query also assigns it a new color tag. Hovering over a query with the mouse pointer reveals a clickable gear icon that opens a configuration window for the selected query. Here, the user can construct the query and assign it a unique title for better identification. This process is illustrated in figure 2.4.

Figure 2.4 Query configuration window



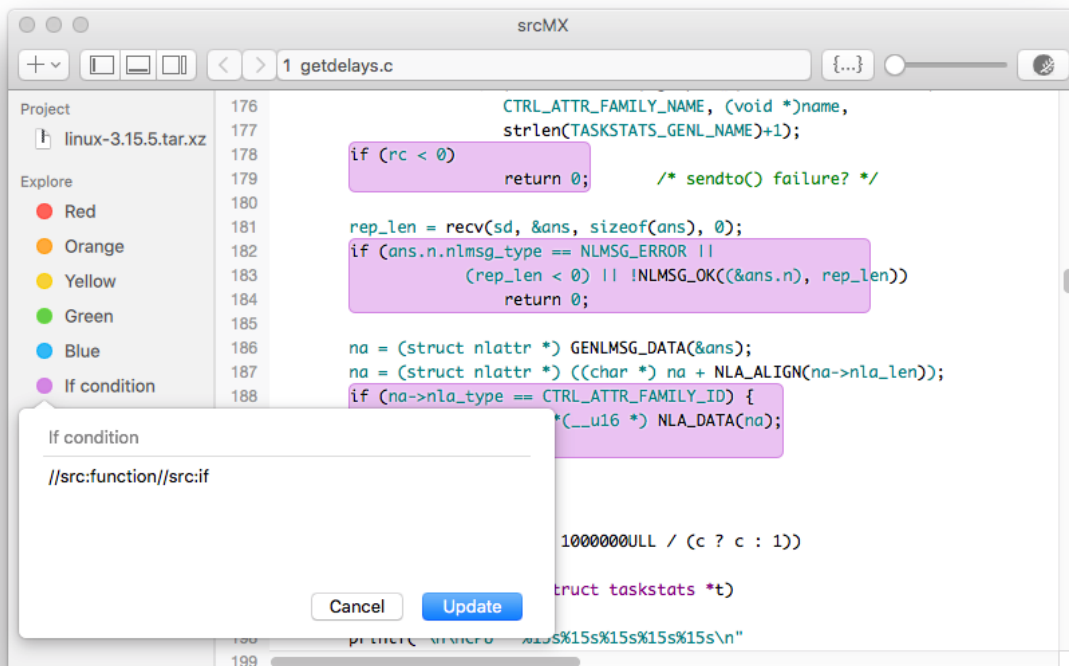
The srcMX application accepts XPath queries, and it supports an autocomplete feature for srcML tag names. As the user types, the candidates are displayed in a dropdown menu. Selecting a tag name with the return key inserts the name. Utilizing this autocomplete feature guarantees that the XPath query is constructed with valid tag names. Consequently, the autocomplete also triggers the query to be executed as each tag name is entered. This allows the user to view a progression of the results as the query is constructed.

The central view in the application contains the file list, which displays all of the files that were parsed and translated from the input source. This list is populated in real time as each source code file is translated to the srcML format by the srcML command-line tool. Here, the user can see the unit number, name, path, language, and lines of code, and parsing time for each file. The user can also select files from this list to display in the code view.

The code view is the most predominant window of the srcMX application. It is also the only window that the user can float and dock using the toolbar buttons. Here, the application dynamically displays source code files in both native and srcML formats. Additionally, the slider in the toolbar above can be used to progressively enable or disable the visibility of srcML tags without reparsing or reloading the current file. The text displayed in the window can also be zoomed in and out using the Command-Plus sign (+) and Command-Minus sign (-) keyboard shortcuts.

The code view also displays query results for the currently selected file. These results are represented by colored background boxes that encapsulate the relevant source code. The code view also displays nested and overlapping query results. An example of a file containing query results is depicted in figure 2.5.

Figure 2.5 Code view with query result



Implementation:

The srcMX application is largely written in C++ using Qt Creator, the official IDE (integrated development environment) for Qt applications. It utilizes the Qt and Qt Quick frameworks. Other technologies that I utilize include HTML, CSS, and JavaScript.

While both Qt and Qt Quick provide support for developing GUI applications, srcMX relies on Qt Quick for most of its graphical elements. The remaining application logic uses Qt-extended C++. This allows the GUI and the logic that defines its dynamic elements to be segregated from the main application logic. However, srcMX was not originally implemented in this way. Rather, the structure of the application went through several revisions before arriving at its current state. Initially, I utilized Qt for all parts of the application.

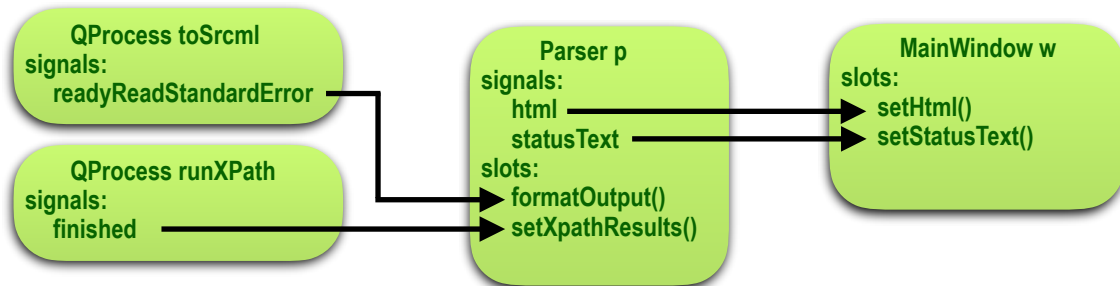
Qt is an application development framework that facilitates the development of cross-platform applications with native GUIs (graphical user interfaces). This framework extends standard C++ with additional features. Before compilation, the MOC (Meta-Object Compiler) preprocessor parses the Qt-extended C++ source files. It then generates standard compliant C++ sources that can be compiled by any standard compliant C++ compiler [5].

GUI applications written with Qt contain graphical control elements known as widgets. These widgets are provided by Qt's QWidget module, which supports rendering to the screen and accepting user input events. With this module, the application's GUI can be defined programmatically. This enables developers to write entire applications directly in C++. Alternatively, developers can define the application's GUI visually by using Qt Designer, Qt Creator's integrated GUI layout and forms designer. With this method,

widgets can be dragged and dropped into a WYSIWYG (what you see is what you get) editor that generates XML layout forms. During compilation, the UIC (User Interface Compiler) converts these XML layout forms into the appropriate C++ code for the GUI.

The Qt signals and slots mechanism facilitates communication between objects. This mechanism provides similar functionality to callbacks. In addition, it includes the added safety of ensuring the type-correctness of arguments. Signals are emitted when events such as user input occur; however, signals can also be emitted programmatically. Connections define which slots are called when the signal is emitted. Then, the appropriate slot function is called according to the connections defined between the objects. Figure 3.2 illustrates this relationship between connected objects in srcMX. During compilation, The remaining C++ source necessary to facilitate these connections is generated by the MOC.

Figure 3.2 Connections of Signals and Slots between objects in srcMX



```
connect(toSrcml, readyReadStandardError, p, formatOutput)
connect(runXPath, finished, p, setXPathResults)
connect(p, html, w, setHtml)
connect(p, statusText, w, setStatusText)
```

Qt's widgets come with many predefined signals and slots. These widgets can also be subclassed if custom signals and slots are needed. Functions defined as slots can also

be called like regular functions. However, if a connection is formed with a class's private slot function, the function can be triggered by a signal from an unrelated object regardless of the function's access specifier. Figure 3.3 depicts the Parser class, which utilizes signals and slots in srcMX.

Figure 3.3 The Parser class declaration utilizing the Signals and Slot mechanism

```
#include <QObject>
#include <QProcess>

class Parser : public QObject
{
    Q_OBJECT
public:
    explicit Parser(QObject *parent = 0);

signals:
    void stdText(const QStringList &text);
    void html(const QString &str1, const QString &str2);
    void statusText(const QString &text);

public slots:
    void src2srcml(const QString &str);
    void text2srcml(const QString &file, const QString &text);
    void setFile(int unit, int loc, const QString &file = "");
    void stopImport();
    void executeXPath(int i, const QString, const QString &xPath);
    void executeQueries(const QString &queries);

private slots:
    void formatOutput();
    void formatTextOutput();
    void formatXPathResults();
    void setXPathResults();

private:
    static QString program;
    static QProcess *toSrcml;
    static QProcess *toSource;
    static QProcess *runXPath;
    int m_unit;
    int m_loc;
    QString m_xml;
    QString m_html;
    bool displayQueries;
};
```

The Parser class is where the srcMX application runs and communicates with the srcML command-line tool in an external program. In this class, the srcml program name and arguments are sent to a QProcess object. The command line tool is started in up to three different processes: toSrcml, toSrc, and runXPath. These three processes allow the srcMX application to utilize several functions of the srcML command-line tool simultaneously. For example, the process toSource can be used to extract a source code file from a srcML archive that the process toSrcml is still in the process of constructing.

As the process toSrcml translates an input file, it prints text to standard error using the verbose argument. The QProcess object emits signals as this text is written. These signals are connected to a slot in the parser object that formats the standard error text for use in srcMX. This text contains information about each source code file, and is used to populate the file list.

When a file is selected in the file view, its srcML is accessed from the srcML archive by the Parser class and displayed in the code view window. To create this window, I subclassed and modified Qt's QWebView widget. However, QWebView's default behavior prevents the XML tags of srcML files from displaying properly. To overcome this, the srcML is parsed, and its tags are wrapped in span elements. Additional HTML is also prepended to the file before being sent to the code view. This allows the application to utilize CSS to manipulate the text in the code view for features like syntax highlighting and marking up query results.

Most of the CSS used in the code view is applied to a user-defined style sheet. This means that the CSS does not need to be reloaded every time a file is loaded, which speeds up browsing time between files. This also allows the visibility of srcML tags to be

enabled or disabled without refreshing the HTML in the web view. This is an important feature, as it prevents the scroll position in the file from being reset when the CSS is updated.

Initially, I experienced difficulty keeping the GUI responsive during parsing. While small files provided little trouble, Importing large archives of source code files would cause the GUI to freeze. This behavior continued even after utilizing external worker threads to format the data provided by the srcml processes. Soon, I realized that the process of updating the file list was causing the application to become unresponsive. The GUI was unable to refresh quickly enough as new entries were appended. In response, I began to investigate other methods of implementing the GUI for the file list. This is when I was introduced to Qt Quick.

The Qt Quick framework is a newer Qt technology that provides an alternative to the Qt widget method of designing GUI applications. It introduces the QtDeclarative module, which provides developers with a new declarative programming approach with QML (Qt Meta-Object Language). QML is a JSON-like language that uses inline javascript expressions for imperative aspects. It not only describes how the user interface of a program looks, but also how it behaves when the user interacts with it. Visual elements known as Quick views are the counterparts to Qt widgets. In addition, Qt Quick provides better support for developing custom user view elements with fluid transitions and effects [5]. Such features are in high demand on the mobile application front. This is accomplished by utilizing hardware acceleration. This made Qt Quick particularly attractive to me, as I was beginning to experience issues with poor responsiveness regarding Qt widgets.

Many of Qt Quick's benefits were immediately evident upon switching to the framework. Overall, the application became much more light weight, requiring approximately 18% of the code that had previously been necessary for GUI implementation. In addition, I discovered that QML provided much more fine-grain control regarding the styling and placement of GUI widgets. Most importantly, the srcMX application remained completely responsive during file parsing.

Unfortunately, switching to Qt Quick also introduced several problems regarding the code view. The Qt Quick view equivalent of Qt's QWebView widget did not support the features necessary to display srcML. I could no longer apply user-defined style sheets to modify the display of srcML. In addition, it appeared that the QML version was optimized for mobile devices, which caused it to behave improperly in desktop applications. In order to circumvent these issues with Qt Quick without sacrificing the functionality of Qt's QWebView widget, I needed a way to integrate both technologies together.

Being a more recent technology, there was not much documentation that explained how to integrate QML in a Qt widget application. One method that I discovered described how to imbed various QML elements within a Qt widget using declarative display views. However, doing so sacrificed the hardware acceleration capabilities of the application. It also introduced undesirable display artifacts. As a result, I was beginning to explore alternate methods for implementing the code view. Luckily, a better solution was made available when a new version of Qt was released. This introduced a widget specifically designed to integrate Qt Quick views inside Qt widget applications. This allowed me to utilize the best features from both previous iterations of my application.

Evaluation:

One primary intent for srcMX was to enable the exploration of source code by performing syntactic and hierarchical searches using XPath queries. This feature can be evaluated by examining the following use cases that a user may encounter in the application. This use case is adapted from an example provided by the official srcML website [6].

Suppose an application developer is using srcMX to assess the documentation of a codebase. The developer would begin by importing the codebase into a blank project within srcMX. After the application has finished parsing and translating the files, the developer can use the query configuration window to begin constructing an XPath query that shows which function declarations that have not been annotated with Doxygen. Using the autocomplete feature, the developer may construct the query in the following manner:

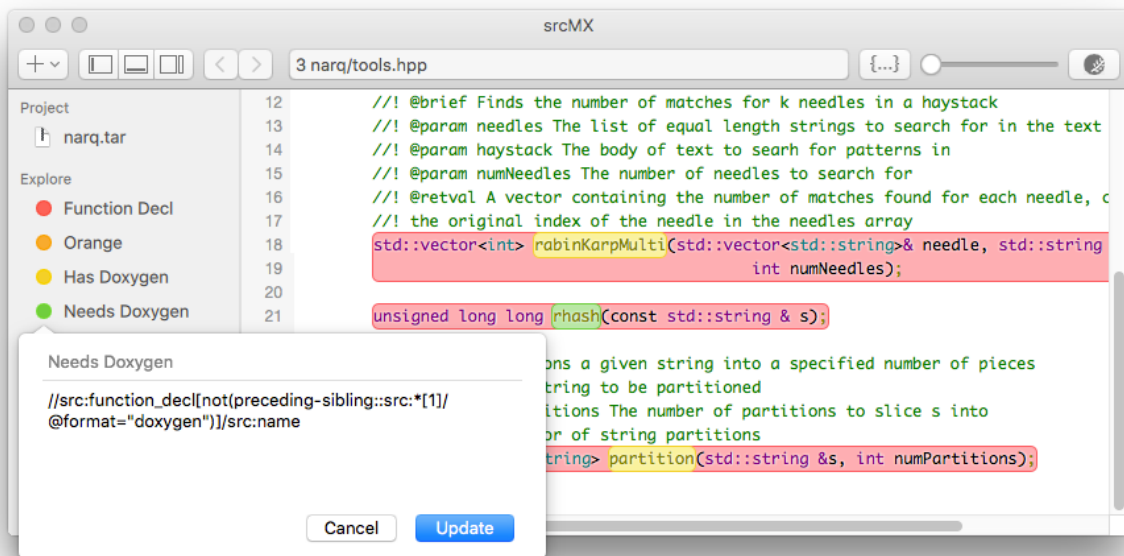
Figure 4.1 Constructing XPath query for functions without Doxygen annotation

```
//src:function_decl  
//src:function_decl/src:name  
//src:function_decl[preceding-sibling::src:*[1]/@format="doxygen"]/src:name  
//src:function_decl[not(preceding-sibling::src:*[1]/@format="doxygen")]/src:name
```

At each stage, srcMX can display the results of the XPath query in the application's code view window. This iterative approach allows the developer to refine the query while receiving visual feedback of the results. First, all function declarations

are highlighted. Next, the results are narrowed to names of the function declarations. Then, XPath's preceding-sibling is used to select sibling nodes of the src:function_decl element. This further narrows results to functions whose first preceding sibling node has the string "doxygen" in the format attribute. Lastly, the previous results are inverted, which returns all functions that are missing Doxygen. Figure 4.2 illustrates this process using multiple nested queries.

Figure 4.1 Constructing XPath query for functions without Doxygen annotation



I also developed srcMX so that it would encourage users to become more familiar with srcML and its many features. My hope is that the user-friendly GUI lowers the barrier to entry for users that are unfamiliar with the features supported by the srcML command-line tool. Instead of offering a help file listing argument and commands, the visual design of the application should communicate its functionality. To achieve this, I ensure that every feature offered by srcMX is represented by at least one visual control. No feature is hidden away as a menu item. Instead, I dynamically show or hide views

depending on the control's context and task the user is carrying out. For example, the file import dialog contains a tab view that lists each supported file type and shows its related controls. This also explicitly communicates to the user what files can be imported. In a similar manner, the control for canceling the import process is also only visible during parsing. The query configuration dialog also exists as a pop-up that is only visible when the user is interacting with it. By showing and hiding these views based on context, I am able to avoid overcrowding the user interface.

Future Work:

Like all software, there will always be a potential for future improvements. Whether it be feature additions, code optimizations, or maintenance, the development process can never be truly complete. However, indefinite development is not sustainable, and there will come time when feature development must end. For srcMX, future work consists of several features that I would still like to implement. For some of these features, groundwork has already been laid for their addition.

Most notably, I wanted to see transformations implemented in srcMX. This was part of my original goal for the application. In fact, I have made progress towards this goal in past iterations. Even the current version has GUI controls dedicated for this purpose. However, the difficulties that I encountered with the core functionality of the application prevented transformations from being fully implemented.

During previous iterations, I had also experimented with an alternate method of accessing files from srcML archives. Currently, files are accessed sequentially by unit number. This poses little trouble when working with several hundred files. However,

there are noticeable delays with large srcML archives like the 38,000 file Linux kernel. Using an experimental build of srcML, I was able to store files in a git archive. This allowed files to be randomly accessed by their hash value. If this feature becomes fully supported by srcML, I believe it would be an excellent addition to srcMX.

Conclusion:

I was able to successfully meet many of my goals for srcMX, and I believe that it has grown into a meaningful addition to the srcML technologies. My application provides developers with a new avenue for exploring and manipulating source code using srcML. In addition, srcMX brings the powerful and robust features offered by srcML to a GUI application for the first time. As a result, a much larger audience can benefit from these features.

Reflecting back on my experiences, I realize just how much I've learned during the development process for srcMX. As my application grew with each iteration, so did my skills as a developer. Despite the difficulties that I encountered during development, I was able to overcome these setbacks using an iterative process that incorporated new technologies. I am proud of what I have accomplished with srcMX. However, the knowledge that I have gained is my true reward. I look forward to exercising this knowledge to continue advancing srcMX with new features in the future.

References:

- [1] Collard, Michael L., Michael J. Decker, and Jonathan I. Maletic. "Lightweight Transformation And Fact Extraction With The srcML Toolkit". 2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation (2011): 173-184. Accessed 8 Aug. 2016.
- [2] "Collaborative Research" *NSF*, National Science Foundation, 10 May 2016, www.nsf.gov/awardsearch/showAward?AWD_ID=1305217. Accessed 8 Aug. 2016.
- [3] "About: What is srcML?" *srcML*, 25 July 2016, ww.srcml.org/index.html. Accessed 29 July 2016.
- [4] "Tutorials: Creating srcML." *srcML*, 25 July 2016, www.srcml.org/tutorials/creating-srcml.html. Accessed 29 July 2016.
- [5] *Qt Documentation*, The Qt Company, 2016, doc.qt.io. Accessed 17 July 2016.
- [6] "Tutorials: Queries." *srcML*, 25 July 2016, www.srcml.org/tutorials/xpath-query.html. Accessed 29 July 2016.