

Version 0.1.0, Project Documentation

Submitted by Charlotte Hirschberger

Table of Contents

[Executive Summary 3](#_Toc490322800)

[User Scenarios………………………………………………………………………………………………………………...4](#_Toc490322801)

[Reports 8](#_Toc490322804)

[System Architecture 9](#_Toc490322805)

[Source Code Structure 10](#_Toc490322806)

[Executables 10](#_Toc490322807)

[Code Architecture 11](#_Toc490322808)

[Database or Data Store 11](#_Toc490322809)

[Stored Procedures 12](#_Toc490322810)

[External Files & Data 13](#_Toc490322818)

[Programming Language | Java 14](#_Toc490322819)

[Project Classes 15](#_Toc490322820)

[Program Start and End Flow 18](#_Toc490322839)

[Summary 20](#_Toc490322840)

[APPENDIX A (DEVELOPER SETUP INSTRUCTIONS) 21](#_Toc490322841)

[Java SE Development Kit 21](#_Toc490322842)

[NetBeans IDE 21](#_Toc490322843)

[GitHub Plugin 22](#_Toc490322844)

[GitHub Account 22](#_Toc490322845)

[Linking GitHub and NetBeans 22](#_Toc490322846)

[Preparing MySQL 25](#_Toc490322847)

[Installing MySQL Server 25](#_Toc490322848)

[Contributing to ChordCommand 25](#_Toc490322849)

[APPENDIX B (CLIENT INSTALLATION INSTRUCTIONS) 27](#_Toc490322850)

[Java SE Runtime Environment 27](#_Toc490322851)

[Installing MySQL Server 27](#_Toc490322852)

[Downloading ChordCommand Files 27](#_Toc490322853)

[APPENDIX C (CLIENT SETUP INSTRUCTIONS) 29](#_Toc490322854)

[Configure MySQL Server 29](#_Toc490322855)

[Prepare the Database 30](#_Toc490322856)

[Create the ChordCommand User 31](#_Toc490322857)

[APPENDIX D (BUILD AND RELEASE PROCESS) 33](#_Toc490322858)

[APPENDIX E (TESTING PLAN) 34](#_Toc490322859)

[Testing Output 34](#_Toc490322860)

[Testing Storage of Preferences 35](#_Toc490322861)

[Retesting Cases TC5 and TC6 36](#_Toc490322862)

[Testing Checksum Methods 37](#_Toc490322863)

[Testing Checksum Methods, cont. 38](#_Toc490322864)

[Overview – Task-Based Testing 39](#_Toc490322865)

[Help & Startup Testing 40](#_Toc490322866)

[Root Layout Testing 41](#_Toc490322867)

[Preferences Testing 42](#_Toc490322868)

[ChordView Testing 43](#_Toc490322869)

# Executive Summary

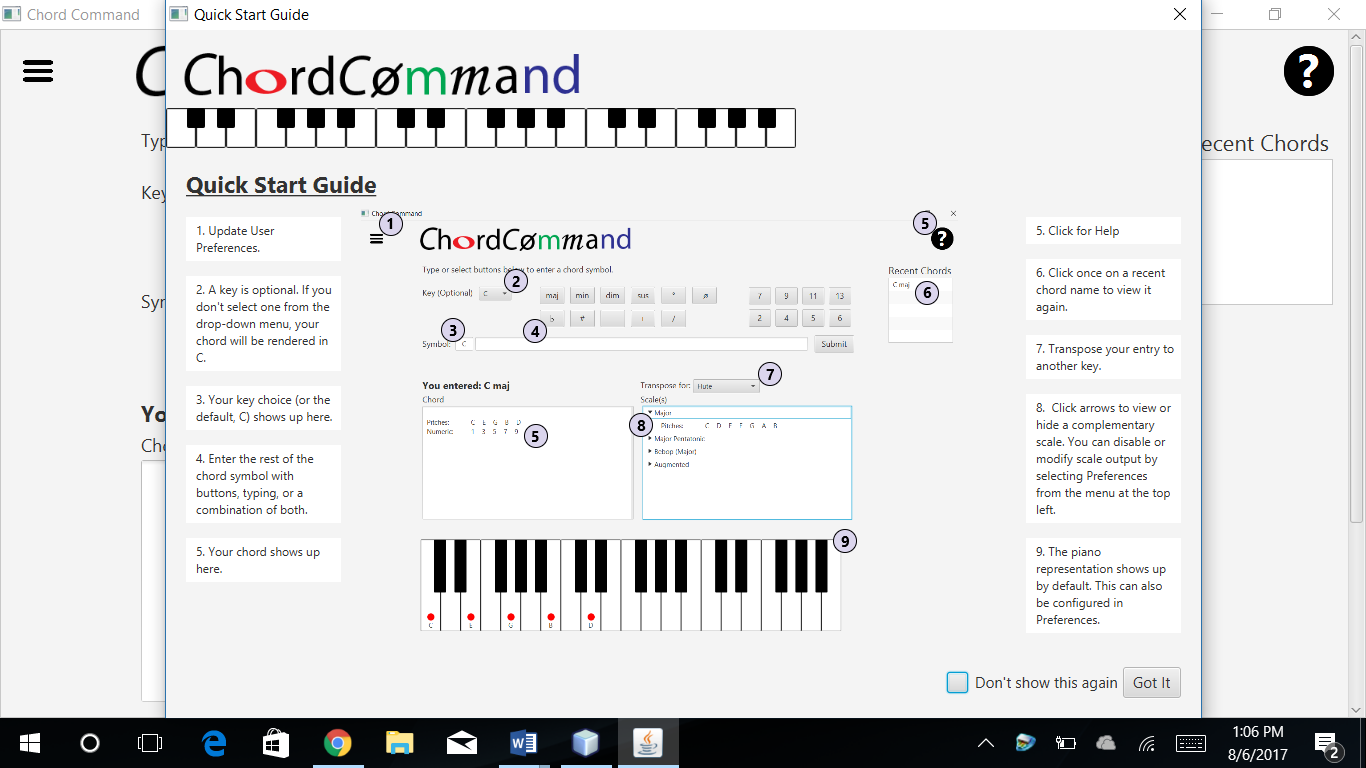
In popular and jazz music, chords are noted using letters, symbols, numbers, and abbreviations. A single chord, like C Major 7, often has multiple representations, like Cmaj7, CM7, and C∆7, which leaves a musician scrambling for a theory book or caroming from one search result to the next. ***ChordCommand*** proposes to solve this problem by putting dozens of chords and scales in one place, with an intuitive user-interface that provides multiple means of entering a chord symbol, like with one’s keyboard and with icons.

For a given chord, ***ChordCommand*** outputs various representations. For example, a Cmin7 entry yields 1-*b*3-5-*b*7; C-E*b*-G-B*b*; and a piano graphic as representations of chord structure and 1-2-*b*3-4-5-*b*6-*b*7-8; C-D-E*b*-F-G-A*b-*B*b*-C; and WHWWHWW as scale representations, with output options configurable by the user.

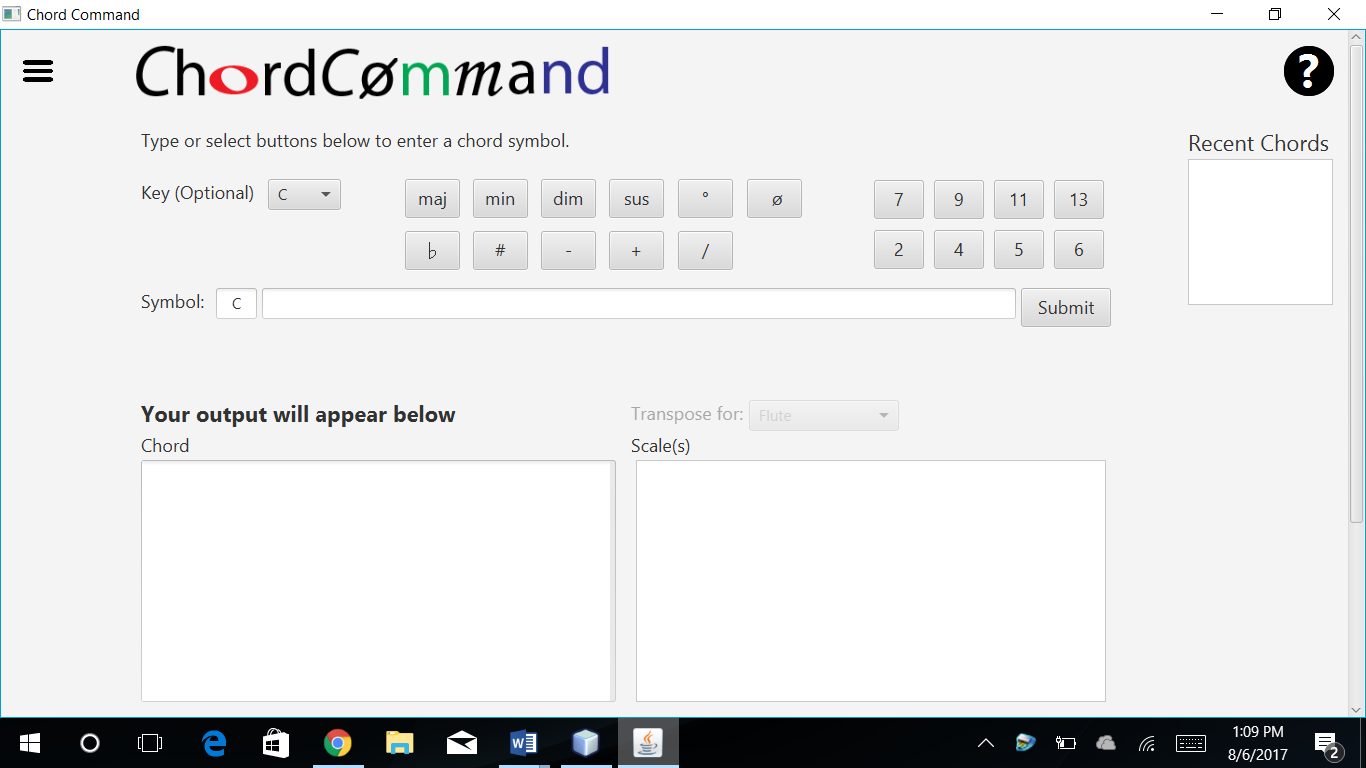
This tool promises to support enthusiasts, professionals, and novices alike. Performers will appreciate the ability to quickly look up any chord that is called for in a gig or rehearsal, while composers can use ***ChordCommand*** to ascertain the structure and sound quality of a chord and verify that it fits into the form they envision. ***ChordCommand*** promises to provide teachers with a tool that guides new and burgeoning musicians, transforming abstract symbols into multiple, helpful visualizations. Students can use ***ChordComman*d** as a handy reference or even as a way of testing their knowledge, and any music lover can develop a richer appreciation of music by using ***ChordCommand***to explore chords from a favorite song or genre.

# Scenario | Using the Help Screen

Joe uses ***ChordCommand*** for the first time. He opens the program from his computer’s start menu, and when the program opens, Joe sees the Help Screen:



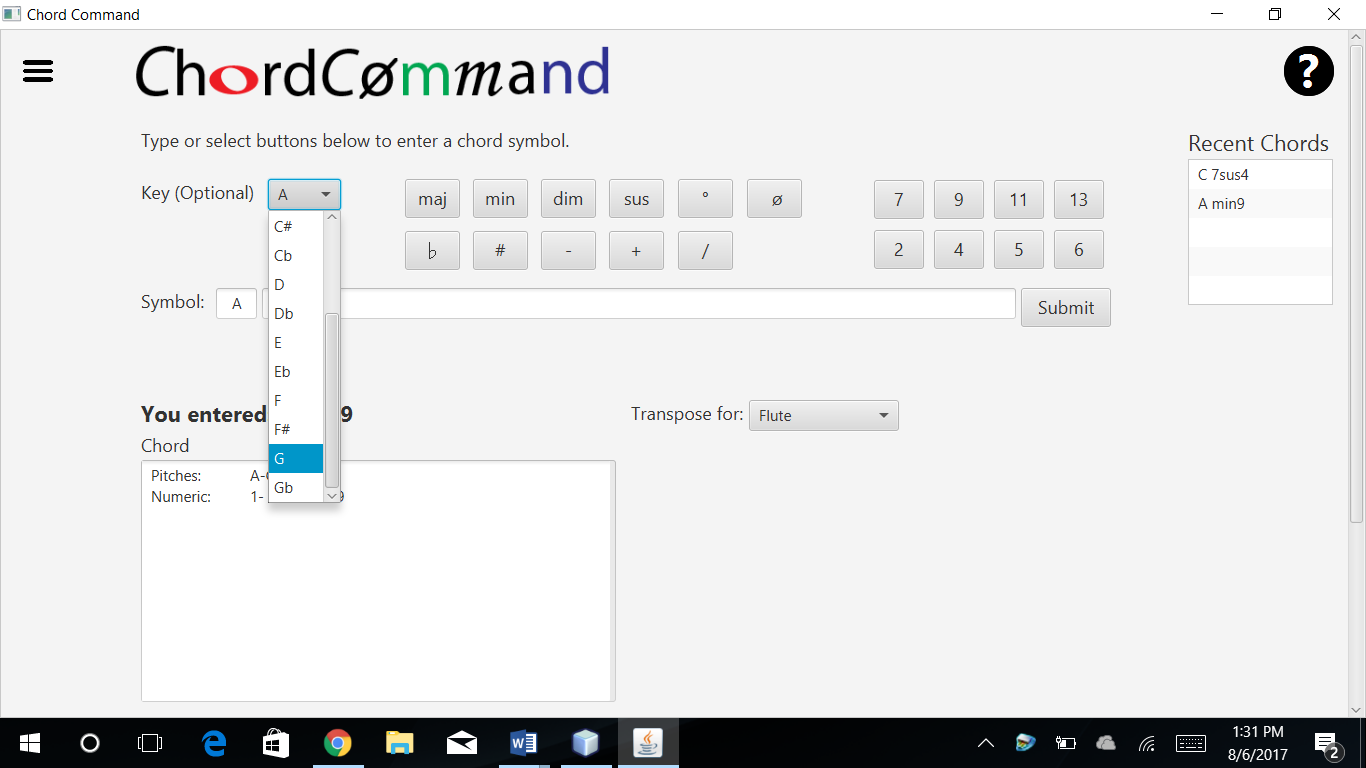
He takes a minute to examine the screen, noticing how the numbers beside the descriptions correspond to parts of the image. When he is done looking at the screen, Joe could choose to click the “Don’t show this again” checkbox, but as a new user, he feels like it might be helpful to see it again on startup. As a result, Joe clicks Got It without selecting the checkbox. The main screen appears:



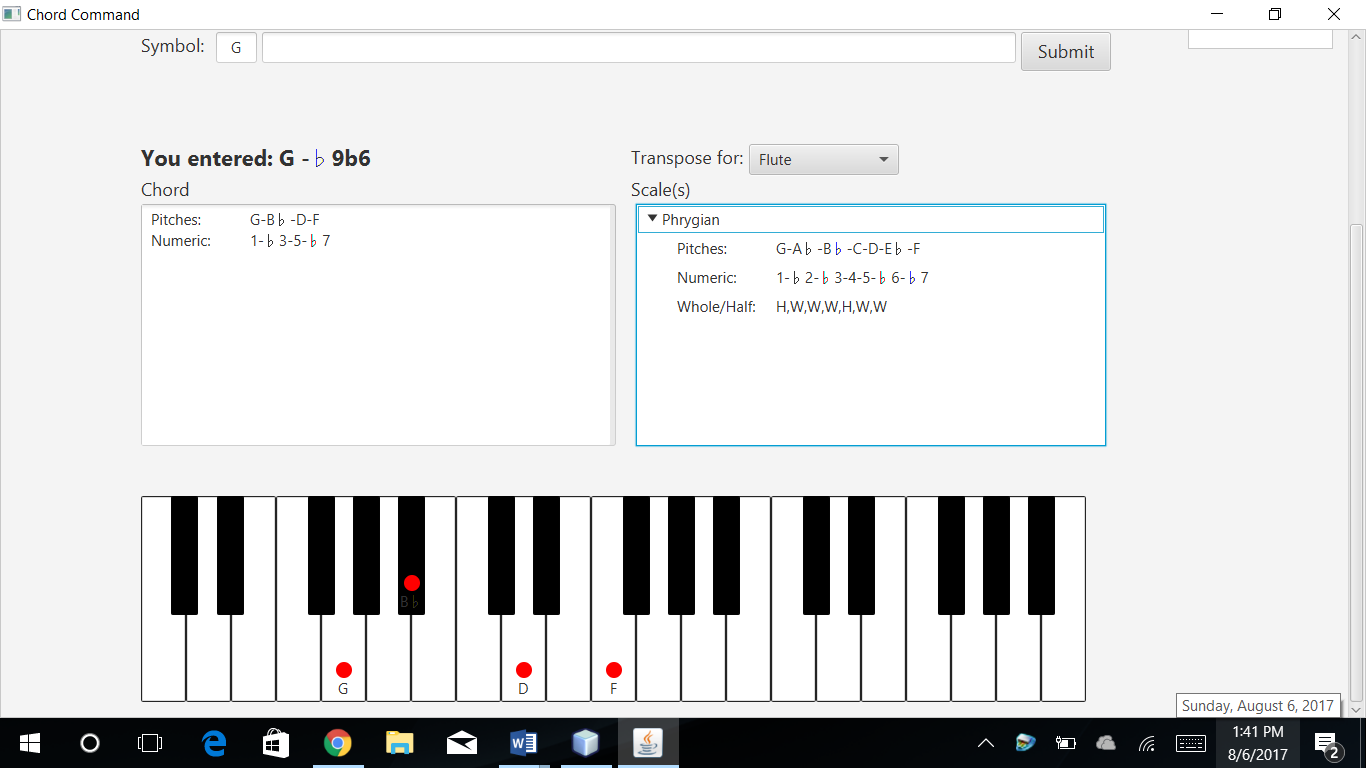
Joe is happy to know that he can see the Help screen again at any time by clicking the question-mark in the top right corner. The Help screen is always available via the icon, even after the user disables it on startup.

# Scenario | Translating a Symbol

Joe is ready to enter a chord symbol. He’s curious about the ‘G -♭9♭6’ chord he saw in some music the other day, so he decides to translate that. He is about to type the “G” when he notices the label “Key” and the box beside it. He uses that to select G:

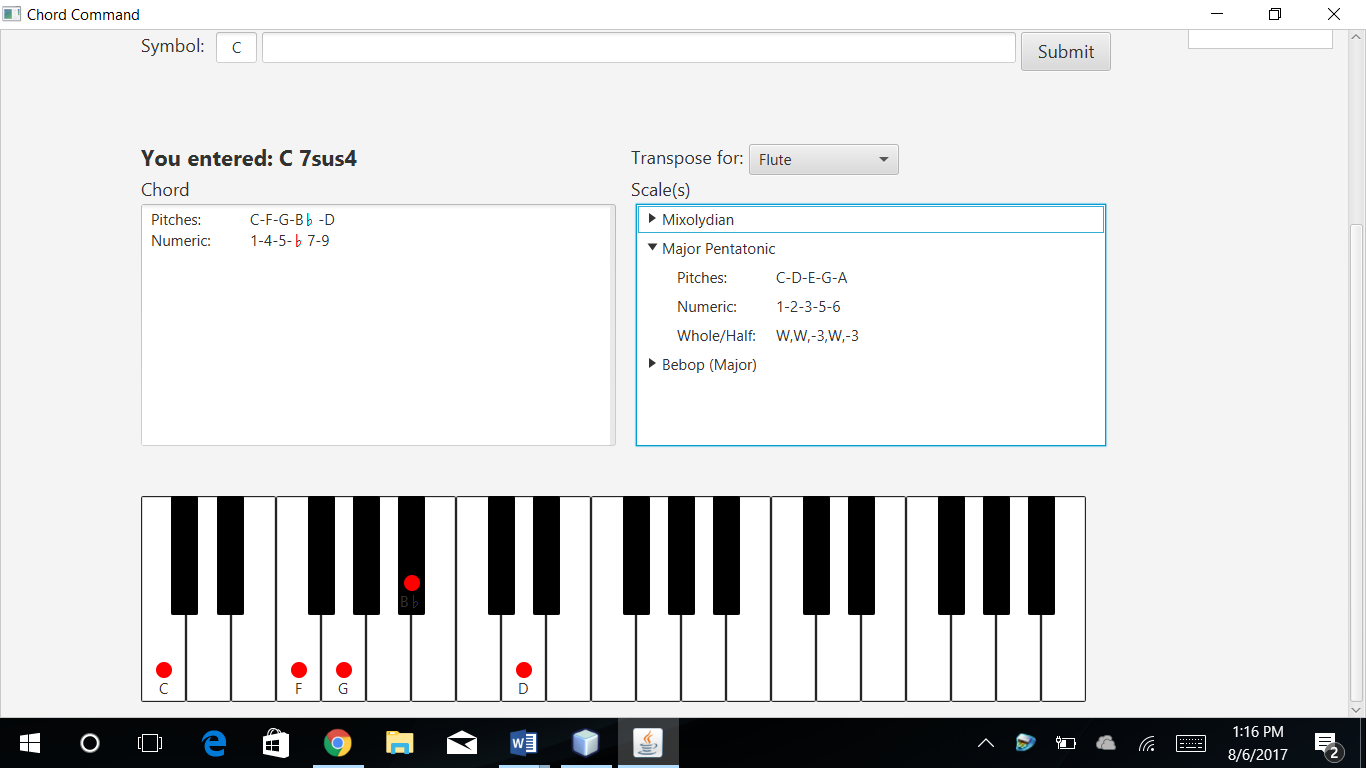


G appears in the little box beside “Symbol:”, and Joe is ready to enter the rest of the chord. The minus sign is easy to type from his keyboard, so he does that instead of using the button. Of course, his keyboard doesn’t have a flat sign, so he decides to use the button below ‘maj’ to enter the flat symbols and sticks with his keyboard for ‘6’ and ‘9’. Joe clicks the Submit button. As he’s doing so, he realizes that he entered ‘b’ for the second flat instead of using the ♭ button. He’s glad to see that his mistake doesn’t matter—***ChordCommand***still manages to display his chord in its output area:

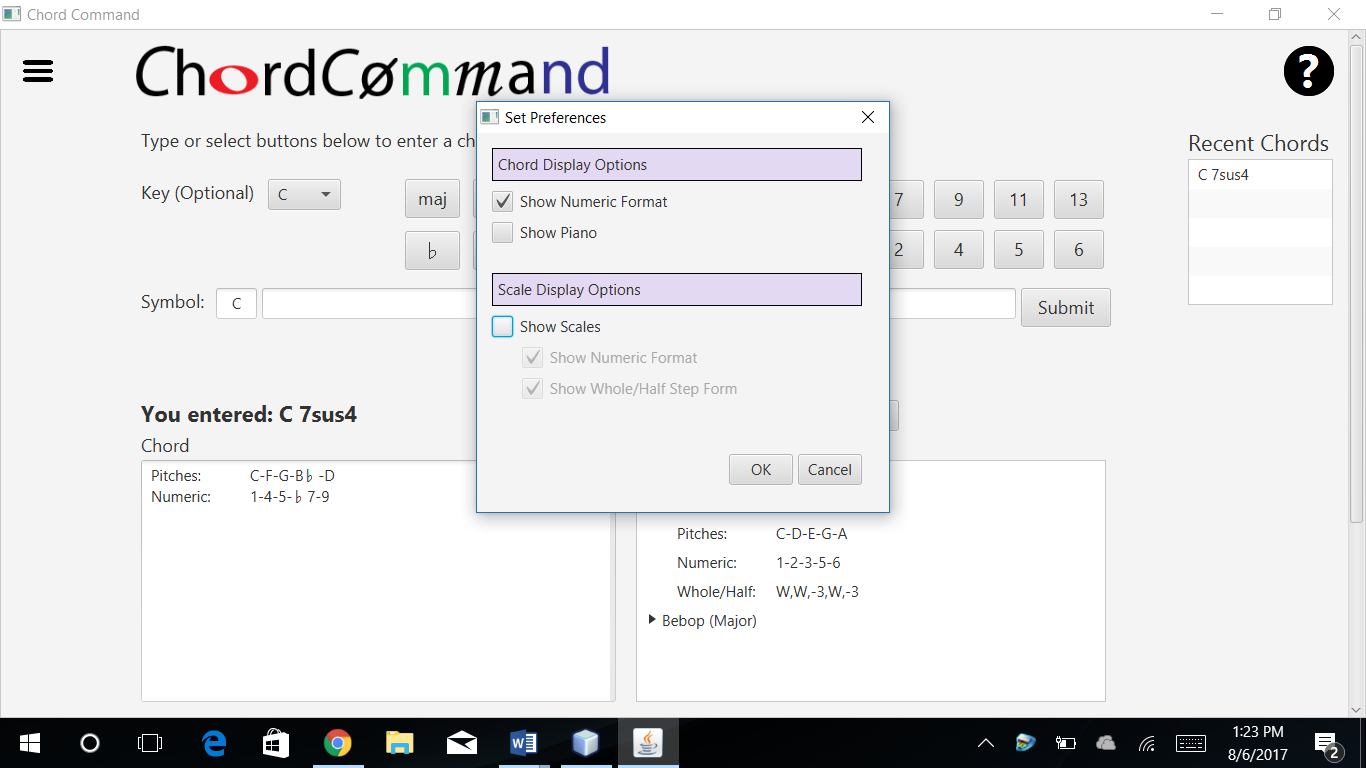


# Scenario | Changing Output Preferences

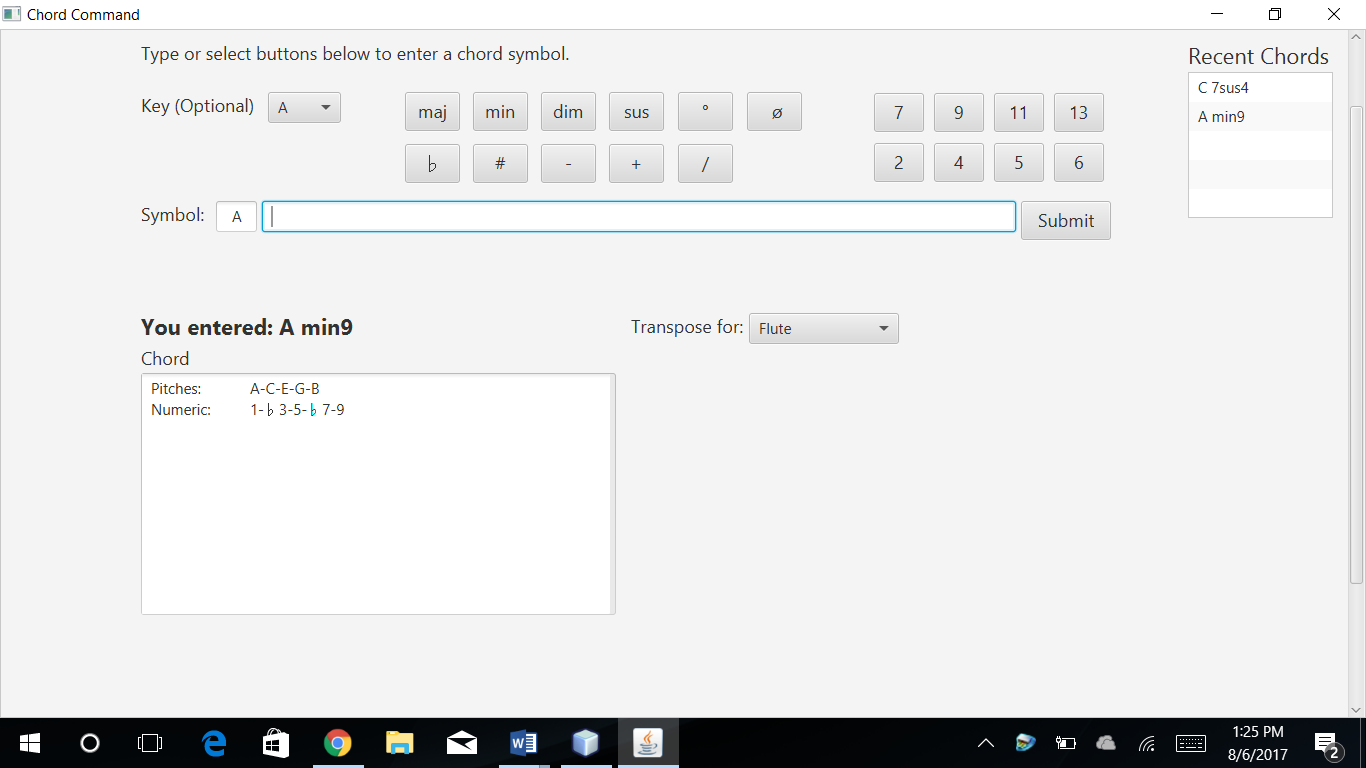
Joe has been using ***ChordCommand*** for a while now, with the default settings that cause chords, scales, and a piano to appear in the output section, as shown below:



As a guitar player, however, Joe really isn’t interested in the Scale and Piano features, so he decides to modify his preferences. He uses the menu icon in the top left corner of ***ChordCommand*** to display a menu, and then selects ‘Preferences’, causing the Set Preferences dialog to appear. Joe quickly notices that he can remove the Piano output by deselecting the check box beside “Show Piano.” Then, he surmises that he can do the same for all scale output by deselecting the check box beside “Show Scales.” Sure enough, when he deselects that check box, all of the other scale options are disabled.



After clicking “OK” and entering another chord symbol, Joe is glad to see that the Scales and Piano sections are removed from the output:



# Reports

8/9/17: This report shows the difference between estimated durations and actual durations over the course of the ***ChordCommand*** project. On average, actual durations were 63% greater than estimated durations. However, these numbers are skewed because some tasks that took ‘days’ only required a couple of hours of effort. For instance, populating tables with data only required 1.25 hours of effort, but that effort was divided over two days in the course of seven.

Together, these findings suggest that estimates should be increased next time, but tasks should also be executed more thoughtfully, no matter how tedious they are. Data entry, for instance, should have been completed in one day or two consecutive ones, instead of being interrupted by class design and stored-procedure creation.



# System Architecture

***ChordCommand’s*** overall structure follows the Model-View-Controller or MVC Architecture, with the addition of data and data-access layers, from which Models are constructed. The MySQL database ‘chorddb’ and .properties files are responsible for persisting data as members of the data layer. The Java Database Connector (JDBC), enhanced by Apache’s Database Connection Pool library, is responsible for coordinating communication between the Java code and MySQL. Meanwhile, the Java Properties class provides the methods that translate the text-based .properties files into objects that Java can interact with.

Moving up in the diagram below, the architecture’s Models and the ChordUtil class are depicted together in the business-logic layer. They are depicted this way because the ChordUtil class verifies that a user’s entry satisfies validation criteria and then employs JDBC/DBCP methods to retrieve the data needed to build MajorKey and MusicStructure Models. Lastly, the Controller classes are depicted apart from the presentation layer, as ***ChordCommand’s*** graphic elements are entirely confined to the View/.fxml objects and are non-functional without Controllers. The Controllers function as the glue between the Views and Models, passing input data from the views to ChordUtil and then initiating the display of data in the Views, after processing in ChordUtil.

This bi-directional flow is typical of the whole architecture. As another example, the business-logic layer passes queries to the data-access layer. The queries are processed in the data layer and then the business-logic layer becomes the receiving-layer, as the data access layer relays the query results to it.

**Data Layer**

chorddb

Database



**Data Access Layer**

Java Database Connector

**Business Logic Layer**

**Presentation Handling Layer**

Controller classes

**Presentation Layer**

Apache Database Connection Pool

Java Properties class

User function interface - View files (.fxml)

.properties files

ChordUtil class

***Instantiates Model objects***

* Validates input
* Builds MusicStructure & MajorKey objects

## Source Code Structure

The following is a summary of the source code directories under version control and their contents:

|  |  |
| --- | --- |
| **Code Directory** | |
| **Directory** | **Usage** |
| ChordCommand/ | Contains *nbproject* and *src* folders |
| ChordCommand/nbproject/ | Contains *project.properties* file, which holds project-specific metadata of use to NetBeans users |
| ChordCommand/src/ | Holds the *chordcommand* package and the ***ChordDbDump.sql*** file, from which database tables and routines are replicated and populated |
| ChordCommand/src/chordcommand | Holds the *ChordCommand* class and the remaining packages: *view, model,* and *util*. *ChordCommand* contains the **main method**, which is responsible for starting execution of the program. The ***dbprops*** and ***userprefs* properties files** and checksums files (.chk) are also located here. |
| ChordCommand/src/chordcommand/view | This directory contains FXML files, controller classes, and image files. The FXML files define the UI components and their layout attributes. The controller classes contain the handlers that allow the user to act on the layout controls. |
| ChordCommand/src/chordcommand/model | This directory contains the classes that ChordCommand’s ChordView and RootLayout controllers use to create objects, whose attributes are displayed on the screen. Those classes include the *MusicStructure* class, its subclasses—*Chord* and *Scale*; *AccidentalMap*; and *MajorKey*. |
| ChordCommand/src/chordcommand/util | This directory contains classes that collect generally applicable methods and/or related modules, with few or no data members. *PropsUtil* holds generic methods that get and set configurations persisted in files. *DBUtil* encapsulates constant database parameters and the methods that act on them. *ChordUtil*’s methods are used to build MusicStructure objects. |
| *Rows in light gray indicate directories that are immediate containers for source code.* | |

# Executables

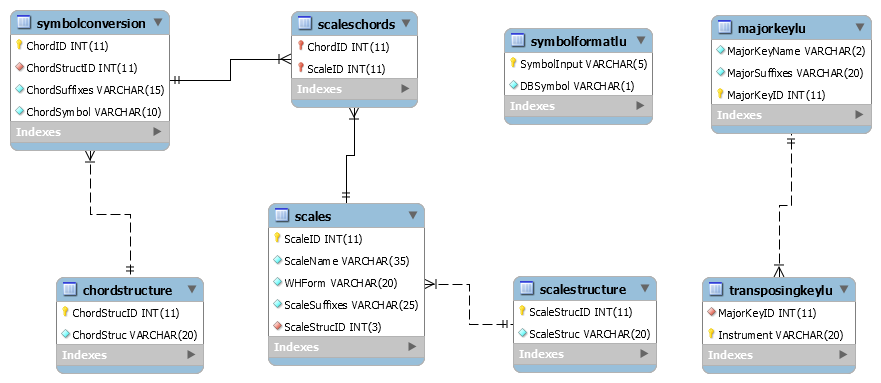
**ChordCommand-0.1.0.exe** is ***ChordCommand’s*** only executable. To obtain it, follow the directions in Appendix C, Client Instructions. When you run this file, an Installation wizard appears. Clicking Install will immediately install ***ChordCommand***’s class files and resources—like images and .properties files—in your Programs directory.When installation completes, you will see the ***ChordCommand*** shortcut in your computer’s Start menu, but before you can use ***ChordCommand***, you must install and configure MySQL, as also described in Appendix C.

# Code Architecture

***ChordCommand*** uses text files to persist configuration settings and MySQL to store the data used to translate symbols. MySQL is an open-source, relational database management system. Its integration with Java is supported via the JDBC API.

## Database or Data Store

The EER diagram for ***ChordCommand***:



As shown, ***ChordCommand*** employs both relational tables and ‘standalone’ tables. The standalone tables serve as look-up resources for the business-logic layer. For instance, ‘min’, ‘mi’, ‘m’, and ‘-‘ can all be used to represent a minor chord, but the database only supports ‘-‘. Using retrieval from **SymbolFormatLU**, the business-logic ascertain that ‘-‘ is the appropriate format for any ‘min’, ‘mi’, or ‘m’ input by the user and then relies on that notation to retrieve minor chords from **SymbolConversion**.

Additionally, ***ChordCommand***’s database employs novel field specifications to facilitate and optimize translation of chord symbols. The **SymbolConversion** and **Scales** tables rely on the integers 1-9 to represent natural pitches; **SymbolConversion, Scales, and MajorKeyLU** rely on the integers -2, -1, 0, and +1 to represent accidentals or “suffixes” that are programmatically applied to pitches.

To begin interpreting a chord symbol, the Java business-logic layer first constructs an array of the pitch names that belong in a given key. For instance, the program uses “D” to determine that D, E, F, G, A, B, and C belong to Dmaj7. The program then retrieves the **MajorKeyLU** record with ID ‘D’ and uses its MajorSuffixes field to construct a key signature. For instance, D major’s key signature is represented as 0,0,1,0,0,0,1 wherein every value but the 3rd and 7th are zero, to indicate that Dmaj7 has two sharps that alter F and C respectively.

The beauty of this scheme is that the calling program can then use values in ScaleStruc and ChordStruc fields to randomly retrieve major-key pitches and suffixes and then compare those suffixes to chord or scale suffixes to determine output. For example, to translate “Dmin7”, the program firsts retrieve pitch names and suffixes for D major, as described above, and stores them in arrays. Next, the program uses the **SymbolConversion** and **ChordStructure** tables to retrieve the ChordStruc and ChordSuffixes for the min7 ChordSymbol, yielding two more comma-delimited strings—1,3,5,7 and 0,-1,0,-1.

The elements at Indices 1, 3, 5, and 7 of both the MajorPitches and MajorSuffixes arrays can now be easily accessed. The MajorSuffixes value at Index 3 is 1; the ChordSuffixes value that corresponds to 3 is -1. Adding these together, ***ChordCommand*** obtains 0 and outputs neither a flat nor a sharp, per the mapping below:

|  |  |
| --- | --- |
| -2 | Double flat |
| -1 | flat |
| 0 | natural |
| 1 | sharp |
| 2 | Double sharp |

## Stored Procedures

Because ***ChordCommand*** uses a sophisticated data scheme to maximize code reuse, the database design includes stored procedures that simplify data entry. These are for use by developers and not via the client, which only interacts with the database for retrieval purposes. Of the stored procedures that follow, *insertScale* and *insertChordRow* are the only two intended for direct use by the database administrator. The remaining stored procedures are used by the insert procedures.

### insertScale

Uses *countCommas* and *isValidStr* routines to insert record into **Scales** table. Performs the following validation tasks:

1. Verifies that WHForm and ScaleSuffixes entries contain the same number of commas as the associated ScaleStruc data.
2. Verifies that WHForm and ScaleSuffixes entries conform to regex criterion.
3. Verifies that hyphens only precede 1’s and 2’s in ScaleSuffixes entries.

### insertChordRow

Uses *countCommas*, *areValidSuffixes*, and *isValidSymbol* routines to insert record into **SymbolConversion**table. Performs the following validation tasks:

1. Verifies that ChordSuffixes entry contains the same number of commas as the associated ChordStruc data.
2. Verifies that a ChordSymbol and ChordSuffixes entries can only valid characters, in a valid arrangement.

### areValidSuffixes

Returns false for entries that contain characters beside 0, 1, 2, and the hyphen. Also rejects entries wherein the hyphen precedes a zero or wherein a hyphen *does not* precede a 2.

### ascends

Not used in the database currently, but will be used in future versions to validate **ChordStructure** and

**ScaleStructure** entries. This procedure verifies that each additional element in a comma-separated string of digits is greater than the previous element and then returns true or false accordingly.

### countCommas

Counts and returns the number of commas in a VARCHAR string.

### isValidStr

Returns true if a testStr argument satisfies the provided regex argument.

### isValidSymbol

Verifies that an entry only contains characters in the following group: -M1-79#b+su ° ø. Also verifies that the symbols ø and ° only appear at the start of the symbol. Returns true if these conditions are satisfied.

External Files & Data

***ChordCommand*** uses text files with the “.properties” extension to store database-connection parameters and user preferences. These files reside in chordcommand package, and moving them will result in the program malfunctioning.

Both properties files are secured using a hashing algorithm to confirm file authenticity. The .chk files in the same directory contain hashes for each file. On startup, ***ChordCommand*** verifies that the hash for each properties file matches the checksum in the corresponding .chk file. If the match fails for the *userprefs.properties* file, the program simply uses default settings that enable all output options. On exit, those default properties overwrite the corrupted content in *userprefs.properties* and a new checksum is persisted.

In contrast, connecting to the database cannot simply be accomplished with a series of identical values. Therefore, if checksum matching fails for the *dbprops.properties* file, an error message appears and ***ChordCommand*** exits, helping ensure that no malicious code is executed.

Programming Language | Java

***ChordCommand*** is implemented in Java. Java is an object-oriented, type-safe language, supported by Oracle (formerly Sun Microsystems). Other aspects that bear mentioning are:

* Platform Independence
* Rich API
* Garbage Collection
* Support for Functional Programming

**Platform Independence**

Oracle bills Java as “write once, run anywhere.” Rather than being compiled directly into code that only one type of OS understands, a Java program is first transformed into “JVM byte code.” Software known as the Java Virtual Machine or JVM then translates the JVM bytecode into a format that the given platform can understand, without additional intervention from the user.

**Rich API**

Java was selected for ***ChordCommand*** primarily due to its rich API, which includes the JavaFX library of graphics and media—an indisputable benefit for any program that requires an interactive and flexible front-end. Additionally, Java’s built-in classes are supplemented by many third-party resources, like Apache libraries and the Spring frameworks. ***ChordCommand*** takes advantage of the Apache library to manage database connections.

**Garbage Collection**

When objects are no longer in use by a Java program, they are automatically eligible for garbage collection, i.e. to have their memory released for other purposes. This is in contrast to a language like C++, wherein the programmer must explicitly deallocate memory in certain situations. Java’s garbage collection lets the programmer focus on innovation, instead of mundane details.

**Support for Functional Programming**

The latest release of Java, version 8, includes support for aspects of functional programming, along with its longstanding support for the object-oriented model. Functional programming allows methods to treat functions as objects that can be passed, returned, and stored, making Java more powerful and versatile than ever before.

Project Classes

Classes within the project are used to abstract re-usable pieces of code. Classes are also used to group related values, known as properties. In Java, classes are indicated with the “.java” extension. The project utilizes these classes:

### ChordCommand.java (chordcommand package)

### This ***ChordCommand*** class contains the project's main method. The ChordCommand object is responsible for initializing the methods that load and save properties, load FXML files and link them to controllers, and populate combo boxes.

### RegionControl.java (chordcommand package)

### This class encapsulates Controller and Region members with accessors and methods that load FXML for the given Region and link it to the Controller member.

### AccidentalMap.java (chordcommand.model package)

### This is essentially a wrapper for a HashMap with Integer ‘key’ and String ‘value’. It defines the correspondence between integers -2 to 2 and accidentals.

### Chord.java (chordcommand.model package)

### Subclass of MusicStructure. The Chord class adds a list of Scale objects, an ID number, and a database-friendly chord symbol to MusicStructure’s members.

### MajorKey.java (chordcommand.model package)

The MajorKey uses a 1-2 letter String to first construct two arrays that represent the MajorKey's pitches and accidentals respectively. The majorPitches array takes the form of 6 letters in the range A-G. The majorSuffixes array takes the form of 6 integers with values between -2 and 2.

MusicStructure.java (chordcommand.model package)

The parent class to Chord and Scale. This defines the characteristics that those musical objects share, including pitches, accidentals (represented by integers), and a major key (which can then be altered by accidentals to produce other sonorities).

Scale.java (chordcommand.model package)

Subclass of MusicStructure. The Scale class simply adds a whForm member and its accessor. The whForm member holds the string that displays the scale’s formation as a series of whole steps (W), half-steps (H), and minor thirds (-3).

AlertSetter.java (chordcommand.util package)

Constructs and displays Alert using given parameters.

ChecksumUtil.java (chordcommand.util package)

This class contains modules for creating and comparing checksums for the purpose of verifying file authenticity.

ChordUtil.java (chordcommand.util package)

This class contains the methods that validate and verify a chord symbol entry and then interact with a database to retrieve data for building Chord and Scale objects. Upon construction of the ChordUtil object, prepared statements are created for use until the program terminates.

DBUtil.java (chordcommand.util package)

The DBUtil class uses Apache's pooling and DBCP libraries to create a connection pool. A limited number of connections remain open and are recycled as necessary, instead of repeatedly establishing new connections, which would drain resources. This class currently contains methods for getting a Data Source, printing Data Source stats, closing a DataSource, and converting a ResultSet to a List.

### PropertiesUtil.java (chordcommand.util package)

The PropertiesUtil class contains methods for creating Properties objects from file content and for persisting Properties in files.

### ChordViewController.java (chordcommand.view package)

The ChordViewController class interacts with the ChordView.fxml file, obtaining input from some controls and setting content in others. This controller is responsible for the buttons, combo box, and text fields where a chord symbol is entered and the panes where output appears, including for the piano.

### Controller.java (chordcommand.view package)

### Base Controller class, from which all of ***ChordCommand***’s other controller classes derive. This allows the derived controller classes to be used in one loadFXML method, despite their otherwise unique compositions.

### HelpViewController.java (chordcommand.view package)

The HelpViewController class interacts with the HelpView.fxml file, obtaining input from a Button and CheckBox and displaying the help Stage when the question-mark icon is clicked. This relies on Controller's empty initialize method.

### KeyMarker.java (chordcommand.view package)

The KeyMarker class creates a Label and Circle for display on a piano key. It derives the KeyMarker's layout attributes from the corresponding Rectangle's position.

### PianoMap.java (chordcommand.view package)

The PianoMap class is simply a wrapper for a HashMap that maps key strings to integers that identify piano keys in the lowest octave. Each value corresponds to multiple keys. Example: B#, C, and Dbb are all keys that map to piano key 1.

### PrefDialogController.java (chordcommand.view package)

The Controller object that is responsible for receiving and processing input from controls in PrefDialog.fxml. Its primary duties are setting and processing each checkbox's SelectedProperty, closing the stage, and handling button clicks.

### RootLayoutController.java (chordcommand.view package)

The RootLayoutController class is responsible for actions on the controls that frame the chord output, including the menu, banner, help icon, and Recent Chords list. The controls are positioned in a GridPane.

Program Start and End Flow

As you can see in the first diagram, much of ***ChordCommand***’s functionality centers on input to the RootLayout and ChordView layouts, both of which reside in the same, main window. To emphasize this, each circuit of functionality that originates and ends at the main screen has arrows of a distinct color.

When the program starts, using the main method in ChordCommand.java, it attempts to load database-connection parameters and the graphic elements that constitute ***ChordCommand***’s screens. If any of these processes fail, control is passed to the Shutdown Hook and the program exits.

If these processes succeed, ***ChordCommand*** checks the showHelp preference property. If that property is set to true, the HelpView appears. Otherwise, the RootLayout and ChordView are immediately loaded and control passes to the user. When the user finishes using the HelpView, he can select the checkbox beside “Don’t show this again” and exit, or he can exit without selecting the checkbox. If the checkbox is selected, the selection is recorded in local memory using the setSinglePref() method in ChordCommand.java. Either way, control soon passes to the main screen, and the program waits for user input.

From there, the user can essentially make one of five choices. The user can exit via the menu or “X”; view help again; change their preferences via the menu; view the About dialog; or type a chord and hit Submit to have it processed and displayed. After all of these decisions, except exiting, control passes back to the main screen and user. Additionally, the non-exit processes can be repeated ad infinitum.

No

Yes

No

No

No

Yes

Yes

Yes

Show Help?

Database credentials & graphics loaded successfully?

Run Shutdown Hook

*This saves preferences and closes connections.*

Call setSinglePref()

Load **RootLayout** and **ChordView**. Wait for input.

User clicks “Got It” or “X”

“Don’t show this again” clicked?

User clicks “X” or selects “Exit” from menu.

User selects “About” from menu

About dialog displayed.

User clicks “OK”

User enters a chord.

**handleSubmit()** subroutine processes entry.

User selects “Preferences” from menu

Load **PrefDialog**.

User makes selections. Clicks “Ok” or “Cancel”.

Is “OK”?

**handleOk( )** sets preferences in local memory.

**handleCancel()** closes PreferencesDialog.

Load **HelpView**

User clicks Help icon.

Summary

In closing, ***ChordCommand*** is designed for ease of use by both clients and developers. A flexible interface gives a user multiple ways to enter a chord. The business-logic accommodates many representations of one chord symbol, and output can be customized through the Preferences dialog.

Likewise, construction of the system architecture, according to the Model-View-Controller scheme, simplifies development and maintenance. Separating components according to their use in presentation (Views), event handling (Controllers), and entity construction (Models) makes code less brittle and more easily modified. The use of .fxml files, coupled with Java’s sophisticated JavaFX GUI Library, means developers can use a visual tool, like Scene Builder, to expedite front-end development. Likewise, stored insert procedures let ***ChordCommand*** use a novel database scheme, with minimal complications for the developer.

Future versions of ***ChordCommand*** are expected to support transcription, guitar representations, mobile use, and more, but this earliest version is the realization of a tool that is comprehensive, easy to use, and will support musicians of all levels and concentrations.

# APPENDIX A (DEVELOPER SETUP INSTRUCTIONS)

These instructions assume that you want to actively contribute to ***ChordCommand***’s development, instead of simply tinkering with the code. If you just want to examine the code, you can obtain the **SourceCode.zip** at <https://github.com/charlotte-hirschberg/ChordCommand/releases>

If you are unfamiliar with Git, it is strongly recommended that you read Atlassian’s Git tutorial to understand the concepts and NetBean’s Git support page for more detailed instructions:

* <https://www.atlassian.com/git/tutorials/what-is-version-control>
* <https://netbeans.org/kb/docs/ide/git.html>

Additionally, these instructions describe the process of contributing to ***ChordCommand*** via NetBean’s Git plugin. Git Shell can also be used, but it won’t be described here. In total, contributing to ***ChordCommand***’s development requires 5 components:

* Java SE Development Kit (JDK)
* IDE with Java support, like NetBeans
* GitHub Account
* GitHub Shell or Plugin
* MySQL Server

The directions for installing MySQL Server are included in this appendix. Details on *configuring* MySQL Server are in Appendix C, Client Setup Instructions. For ***ChordCommand*** to function in both client and development environments, configuration is necessary.

### Java SE Development Kit

1. Navigate to Oracle’s JDK downloads page: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
2. Click the circle next to “Accept License Agreement” and then select the appropriate download for your system. ***ChordCommand*** is currently optimized for Windows, but development efforts for any OS are welcome.
3. You’ll see the download start at the bottom of your browser. If it doesn’t open automatically when complete, click the up arrow beside the download name and select “Open.” If you see a “Do you want to allow the following program to make changes to this computer?” message, select “Yes.”
4. Follow the instructions on the screen to complete the installation.

### NetBeans IDE

Other IDEs, or Integrated Development Environments, that support Java exist, most notably Eclipse. You can use any IDE with Java support, but the instructions here describe the process with NetBeans:

1. Navigate to the NetBeans downloads page at <https://netbeans.org/downloads/>
2. Select your Language and Platform from the dropdown menus at the top right and then click the Download button for Java SE.
3. A new page will open, and your download should start automatically. When the download completes, open it and select “Yes” if asked “Do you want to allow the following program to make changes to this computer?”
4. Follow the instructions on the screen to complete the installation.

### GitHub Plugin

Again, everything you can accomplish (and more) with NetBean’s GitHub Plugin can be accomplished via the Git Shell. However, if you are new to Git or the command line, the Plugin offers a much more user-friendly experience.

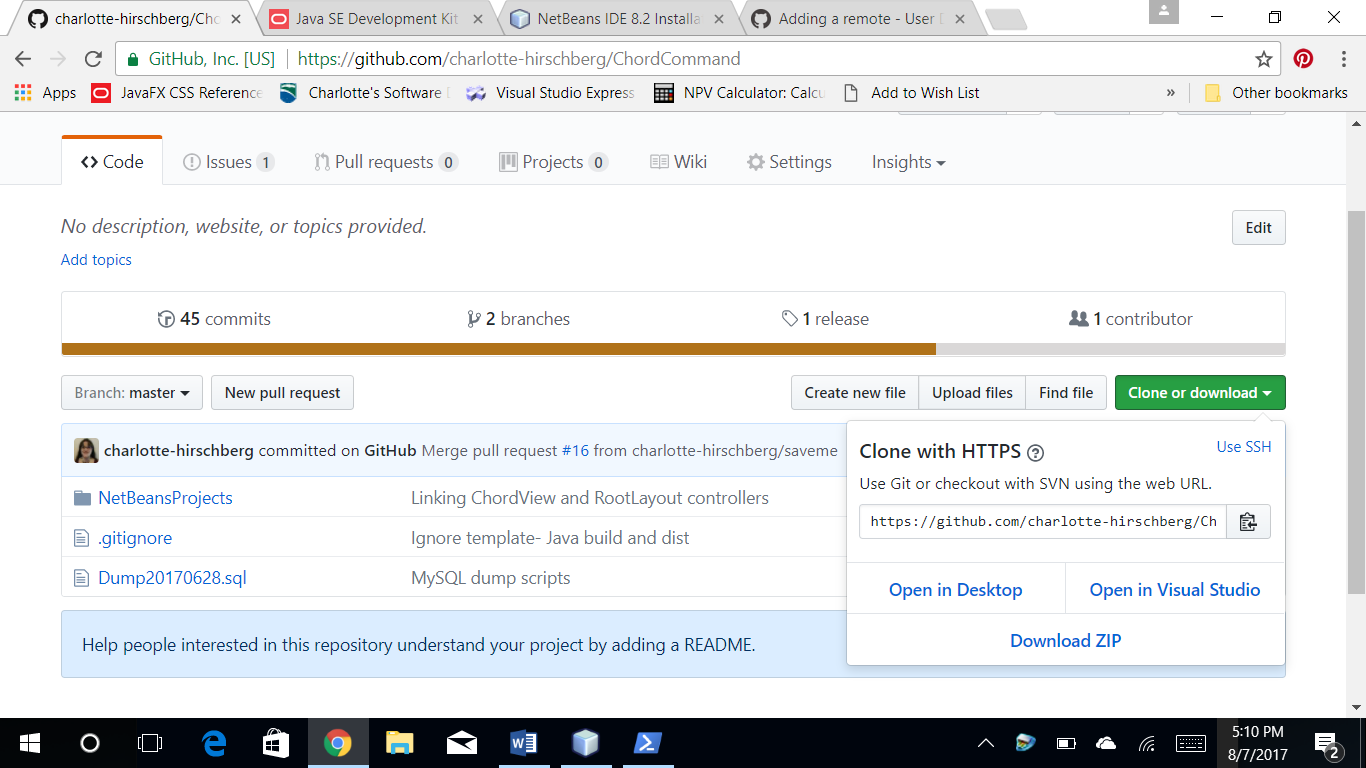
1. Open NetBeans and select “Plugins” from the Tools menu.
2. Switch to the “Available Plugins” tab, click the check box next to “Git”, and select Install. Follow the instructions on the screen to complete the installation.

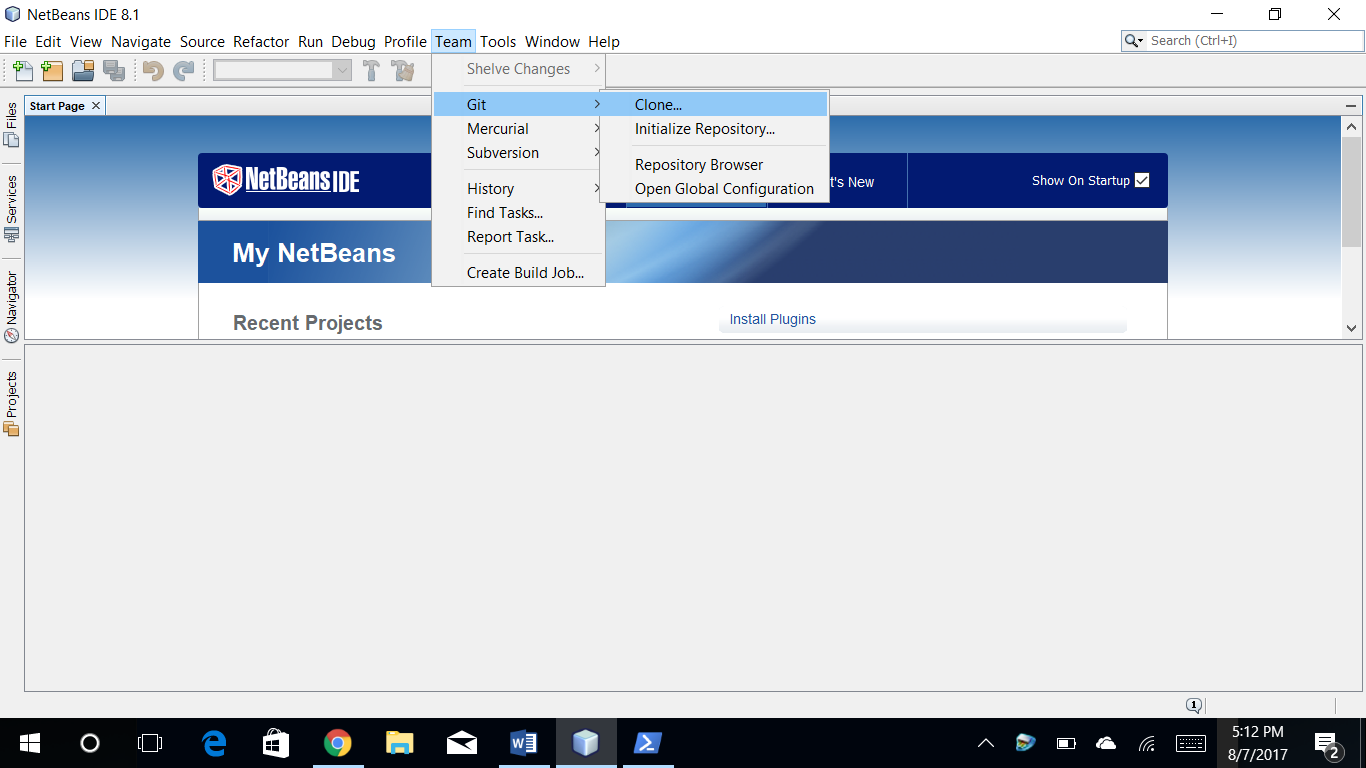
### GitHub Account

1. If you don’t have a GitHub account, register for one at <https://github.com/join>
2. Navigate to the ***ChordCommand*** repository at <https://github.com/charlotte-hirschberg/ChordCommand>. To push code to the repository for approval and incorporation by the ***ChordCommand*** team, you need to be an official Collaborator. Email [charlotte.hirschberg@mymail.champlain.edu](mailto:charlotte.hirschberg@mymail.champlain.edu) to be added as a Collaborator.

### Linking GitHub and NetBeans

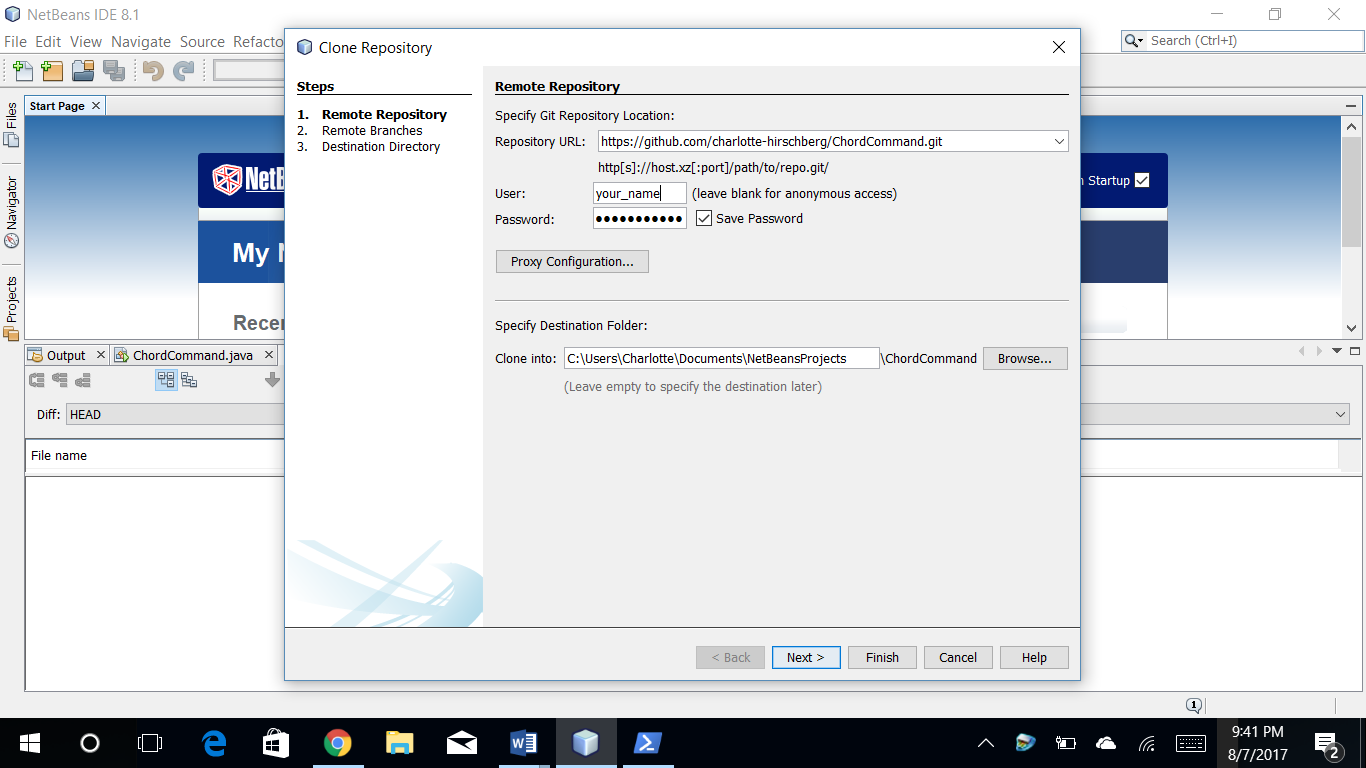
1. From the ***ChordCommand*** repository, click the green “Clone or download” button and then select the picture of a clipboard to copy the repository URL to your computer’s clipboard.



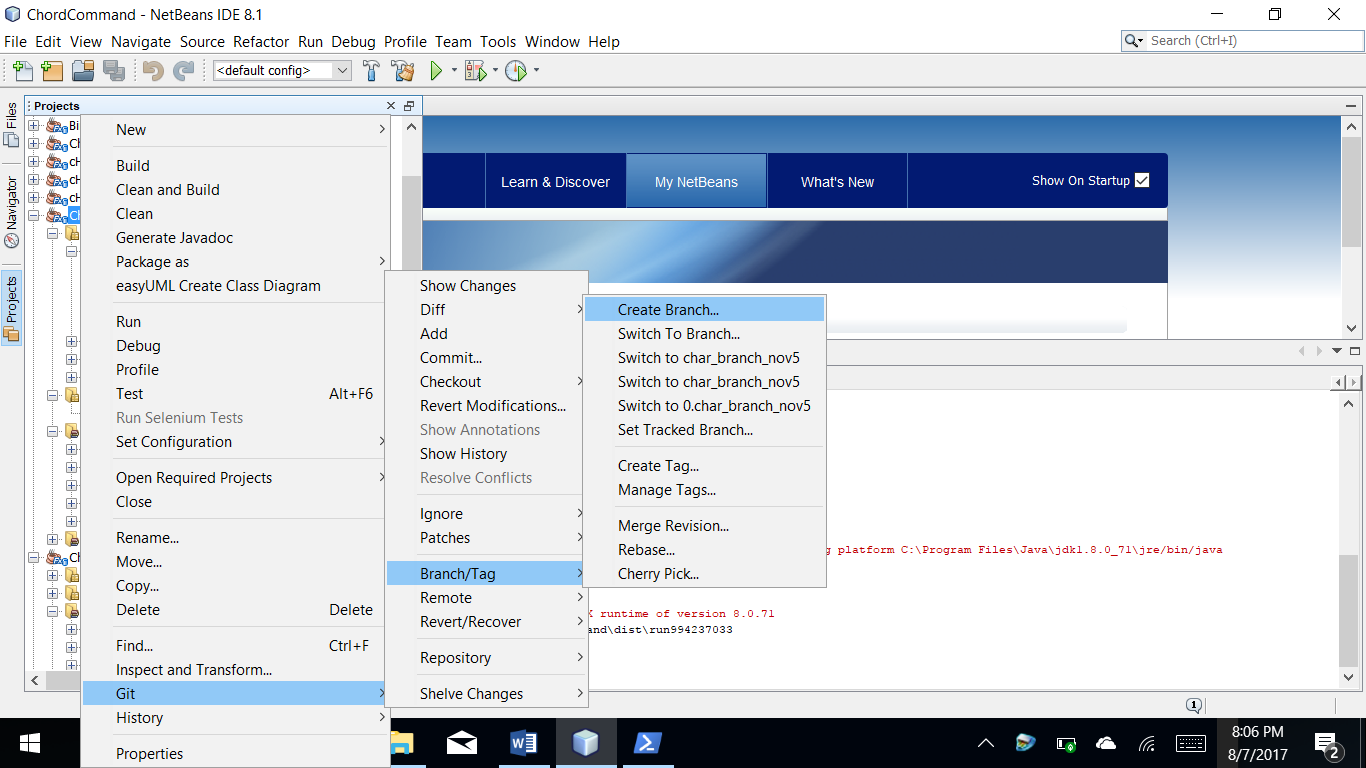
1. Open NetBeans. From the “Team” menu, access Git > Clone.

(cont. on next page)

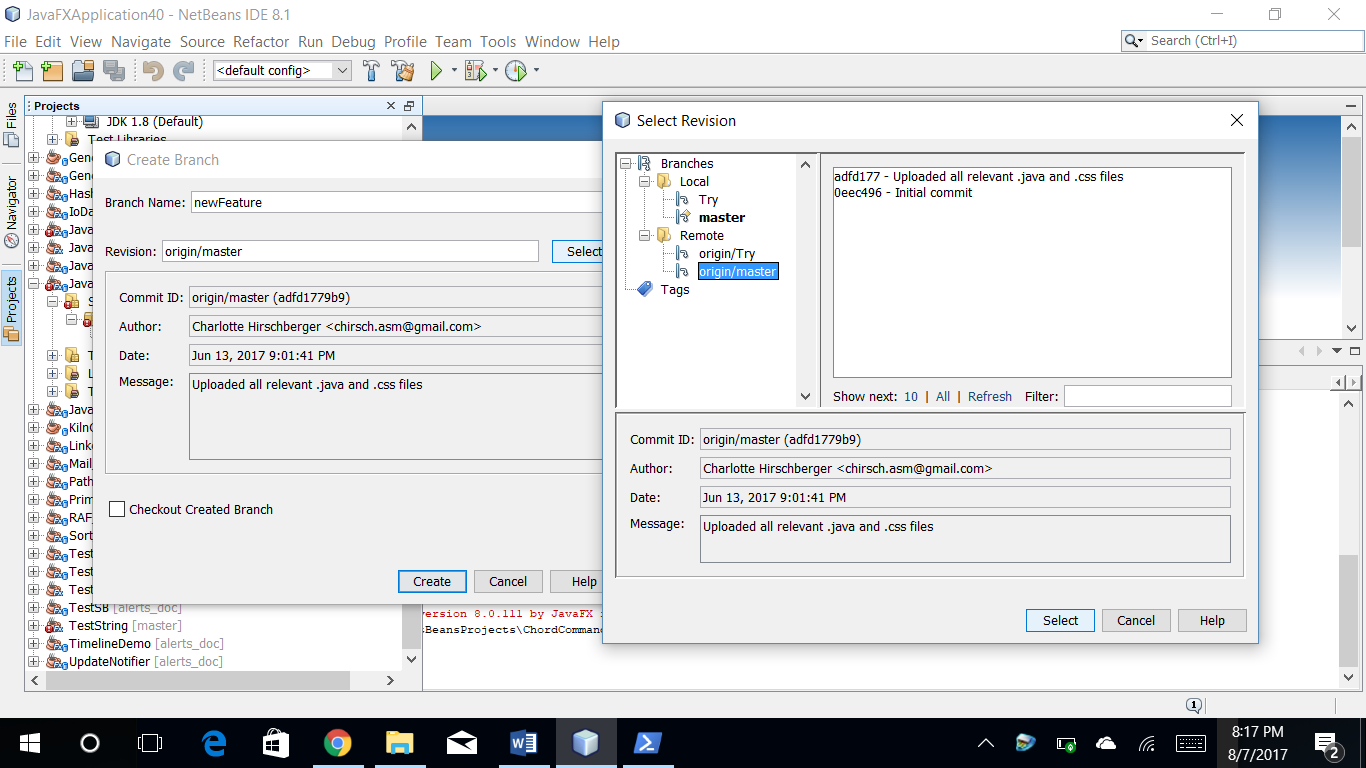
1. In the “Clone Repository” dialog, use CTRL+V to paste the ***ChordCommand*** repo URL in “Repository URL”. Enter your GitHub username and password and then choose a new directory for the local files, if you like.

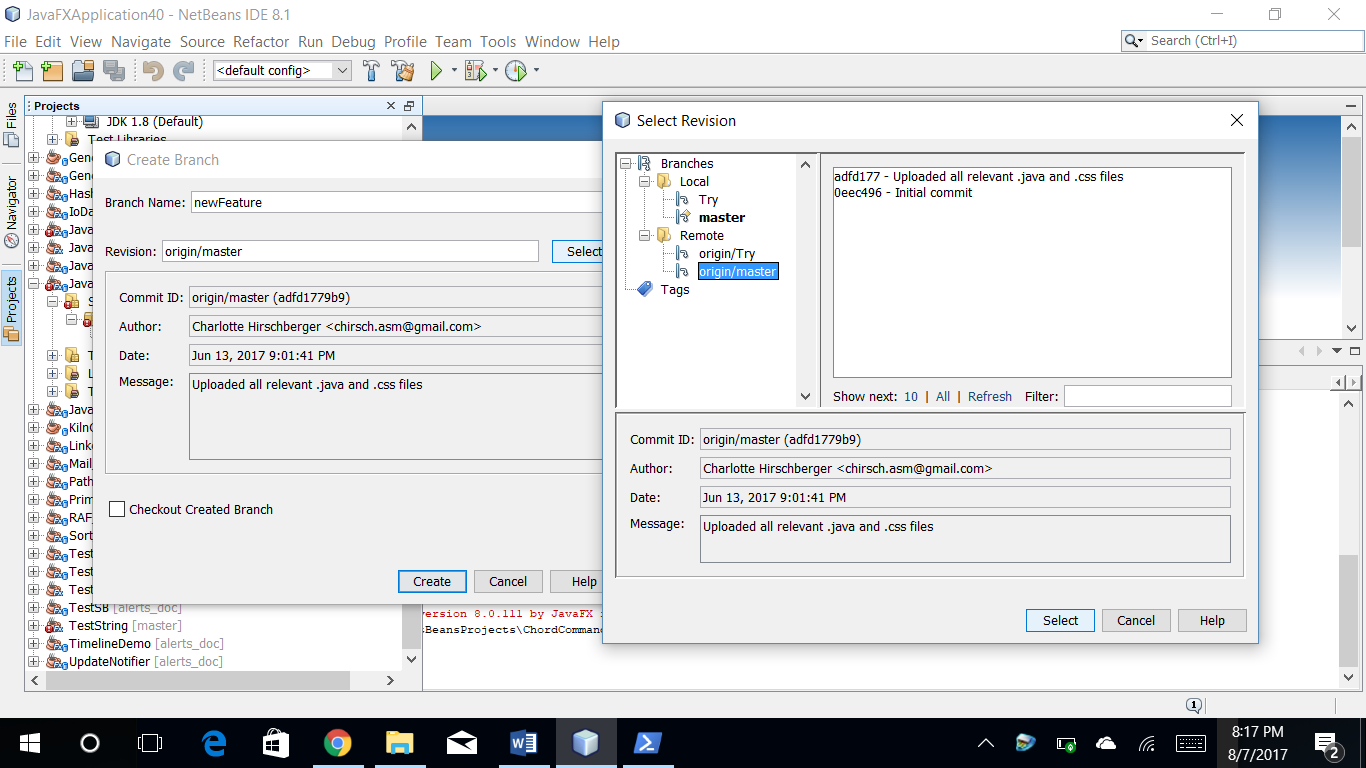


1. Click “Next” and, on the Remote Branches page, select the check box beside “master” if it isn’t already selected.
2. Click “Next” again to navigate to the “Destination Directory” page. Choose a different “Parent Directory” name if you like, and make sure the check box beside “Scan for NetBeans Projects after Clone” is selected. Leave the other fields alone.
3. Click “Finish” and NetBeans will ask if you want to open the project. Select “Open Project”. Now you’ll be able to double-click file names in the Project window and edit code.
4. Now make a branch where your modifications will reside. To make a new branch in NetBeans, right-click any object in the ChordCommand directory. Navigate to Git > Branch/Tag > Create Branch as shown below.



1. In the Create Branch dialog, type your choice of Branch Name, select the check box beside “Checkout Created Branch”, and then click the “Select” button to specify a revision. These elements are outlined in red below.



1. In the Select Revision dialog, make sure you open Branches > Remote and then click “origin/master”. Your screen should look similar to the one below, with origin/master highlighted in blue. Verify that it does and then click “Select.” Click “Create” to close the remaining dialog. 

### Preparing MySQL

Use the instructions in the following section to Install MySQL Server, and then consult the listed sections in Appendix C, with changes as noted:

* Configure MySQL Server—as directed
* Prepare the Database—in Step 3, you’ll still be importing from the ChordDbDump.sql file, but you’ll locate the file in the ChordCommand directory that you specified as the destination in Linking GitHub and NetBeans Step 3
* Create the ChordCommand User—as directed

### Installing MySQL Server

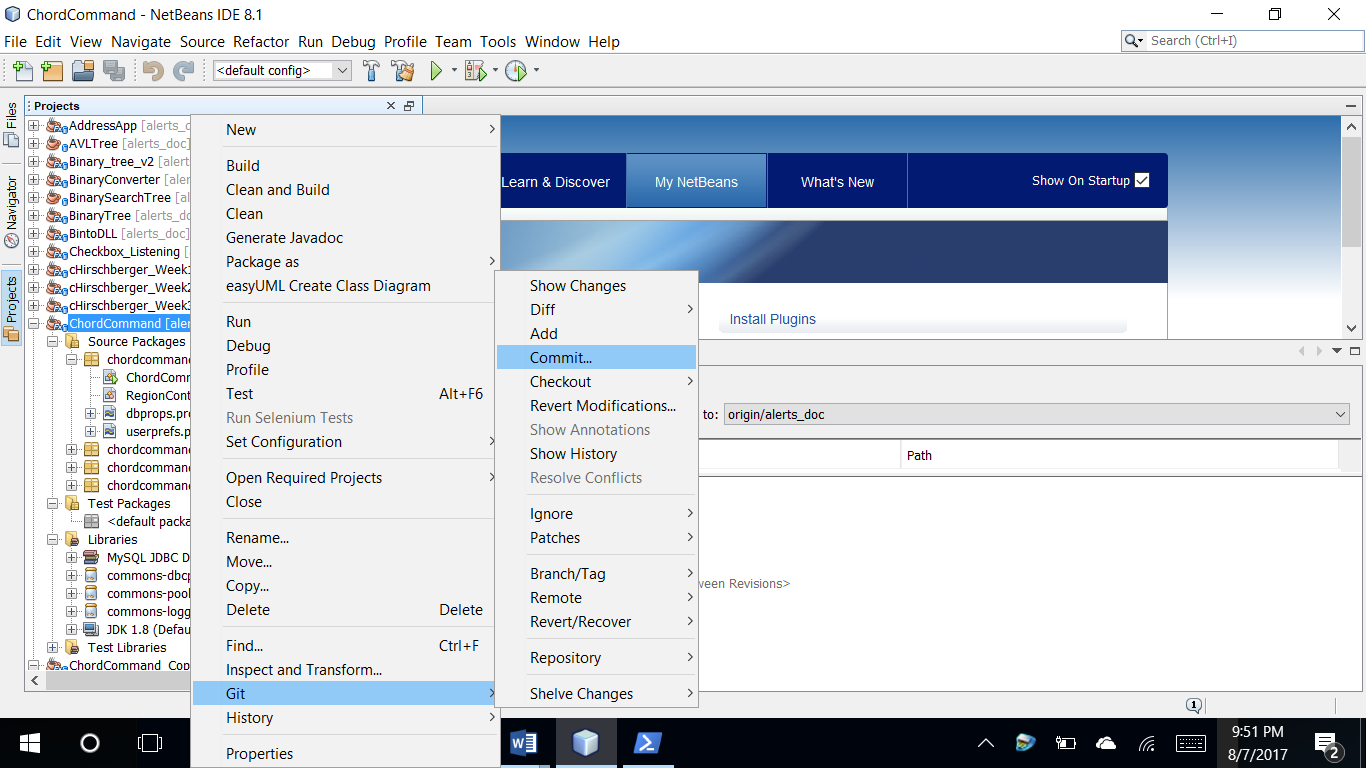
Along with the server, we recommend installing MySQL Workbench to simplify the process of working with data and connections. MySQL Installer conviently packages these components together, along with documentation and other handy development tools. Therefore, we’ll download and use the Installer in these next steps:

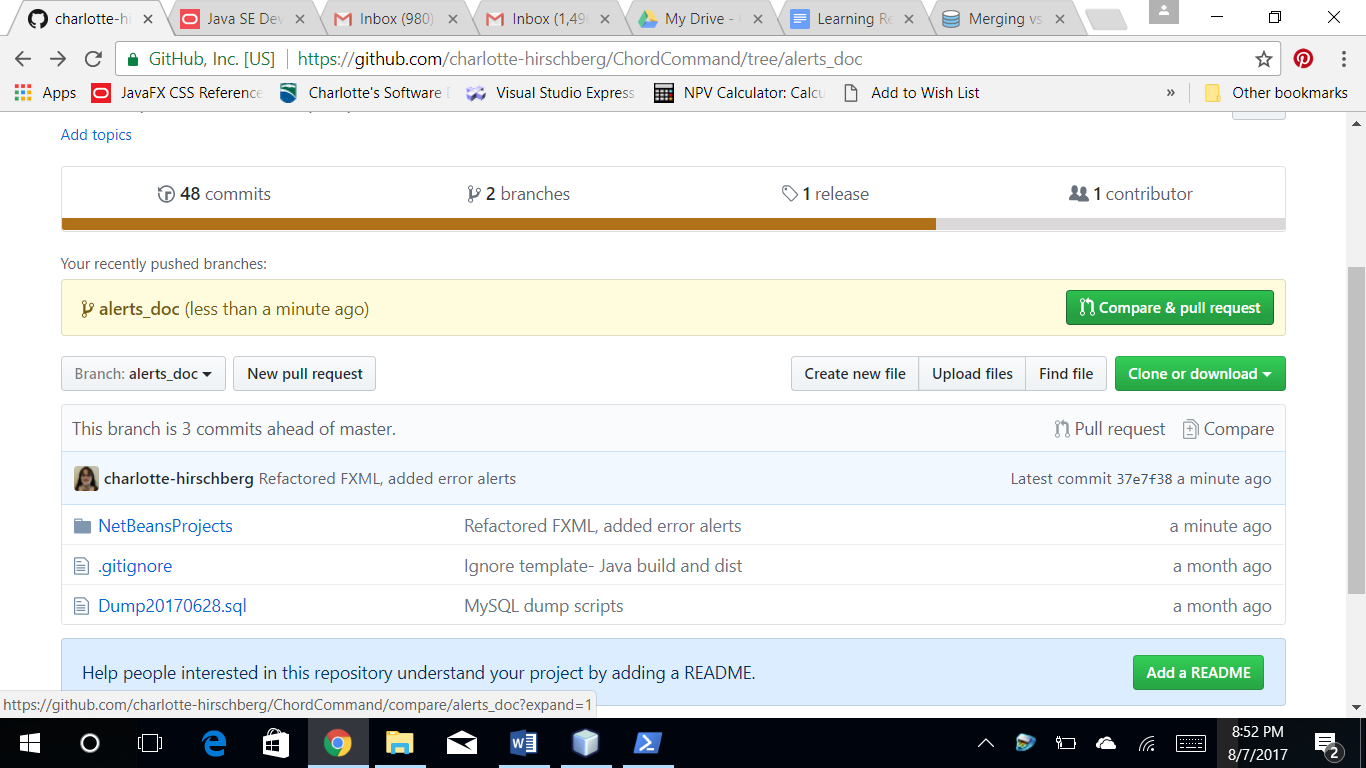
1. Navigate to the downloads page for MySQL Installer: <https://dev.mysql.com/downloads/installer/>
2. Scroll to the bottom of the page and select a download. Per the MySQL documentation, “if you have an online connection while running the MySQL Installer, choose the mysql-installer-**web**-community file.” Otherwise, select the second download.
3. You’ll be prompted to “Login” or “Sign Up” but you can ignore these options by scrolling to the bottom of the page and choosing “No thanks, just start my download.”
4. As with the JRE, you’ll see the download start at the bottom of your browser. If it doesn’t open automatically when complete, click the up arrow beside the download name and select “Open.” If you see a “Do you want to allow the following program to make changes to this computer?” message, select “Yes.”
5. Follow the instructions on the screen to complete the installation.

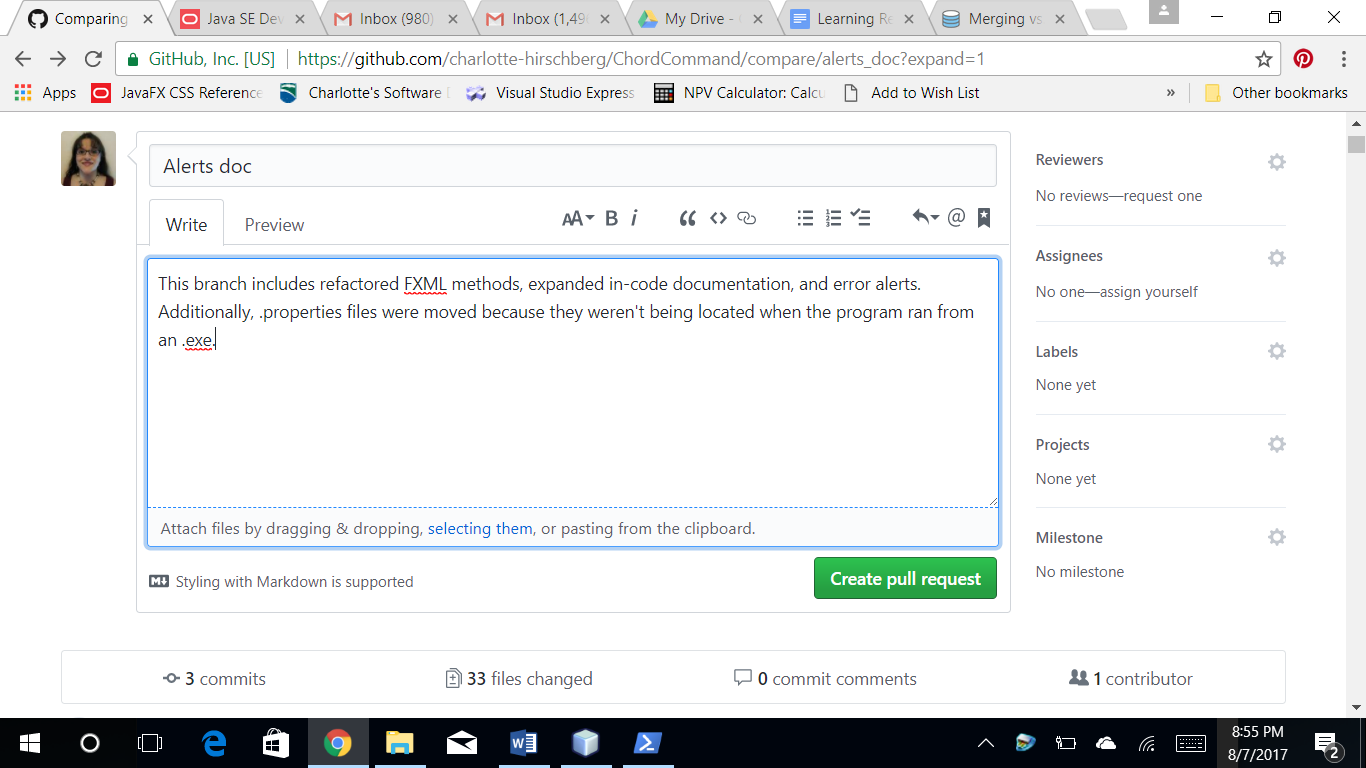
### Contributing to ChordCommand

At this point, you can edit ***ChordCommand*,** and it will be fully functional when you run the project in NetBeans. When you’re satisfied with your additions, you need to commit and push your changes, as described below:

1. In the NetBeans Projects window, right-click the project name and access Git > Commit.



1. In the window that opens, you’ll see a list of project files that have been added or modified. Type a concise but specific “Commit Message” that explains your changes and then hit “Commit.”
2. Once again, right-click the project name. Access Git > Remote > Push.
3. In the window that opens, verify that the “Configured Git Repository Location” is <https://github.com/charlotte-hirschberg/ChordCommand> and then select “Next”.
4. On the “Select Local Branches” page, select the *branch you created*, not master. Click “Next”. Click “Finish” on the “Update Local References” page.
5. When asked if you want to “Set Up Remote Tracking?” choose “Yes”.
6. Via your Web browser, navigate back to the ***ChordCommand*** repo. Expand the Branch drop-down, and you’ll see your branch. Click to select it.
7. Click the big green “Compare & pull request” button in the right corner above the project files. 
8. In the page that appears, add some comments to fully explain your changes and then click “Create pull request.”



You’re done! On the next page, GitHub might tell you that merging can be completed automatically, but please do not merge the pull request yourself. Let others test and review your code, and a member of ***ChordCommand***’s core development team will approve and perform the merge. Thank you!

# APPENDIX B (CLIENT INSTALLATION INSTRUCTIONS)

This appendix will walk you through the steps required to install the components that ***ChordCommand*** requires for use on a Windows machine. You will need to install the following components:

* Java SE Runtime Environment, aka “JRE”
* MySQL Server & Workbench
* ChordCommand

Once you’ve installed these components, please consult the section that follows, Appendix C, for information on configuring these components, so that they are able to work together.

### Java SE Runtime Environment

If Java is not installed on the machine, follow these steps:

1. Navigate to the JRE downloads at <http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>
2. Click the circle next to “Accept License Agreement” and then select a Windows download ending in “.exe”. If you’re not sure what type of system you have (x86 or x64), you can find this information in Settings>>System>>About.
3. You’ll see the download start at the bottom of your browser. If it doesn’t open automatically when complete, click the up arrow beside the download name and select “Open.” If you see a “Do you want to allow the following program to make changes to this computer?” message, select “Yes.”
4. Follow the instructions on the screen to complete your download. The installation wizard will notify you if it finds another version of Java on your machine.

### Installing MySQL Server

Configuring MySQL Server is a bit more complicated. Along with the server, we recommend installing MySQL Workbench to simplify the process. MySQL Installer conveniently packages these components together, so you’ll download and use the Installer in these next steps:

1. Navigate to the downloads page for MySQL Installer: <https://dev.mysql.com/downloads/installer/>
2. Scroll to the bottom of the page and select a download. Per the MySQL documentation, “if you have an online connection while running the MySQL Installer, choose the mysql-installer-**web**-community file.” Otherwise, select the second download.
3. You’ll be prompted to “Login” or “Sign Up” but you can ignore these options by scrolling to the bottom of the page and choosing “No thanks, just start my download.”
4. As with the JRE, you’ll see the download start at the bottom of your browser. If it doesn’t open automatically when complete, click the up arrow beside the download name and select “Open.” If you see a “Do you want to allow the following program to make changes to this computer?” message, select “Yes.”
5. Follow the instructions on the screen to complete the installation. When installation completes, you’ll see a MySQL folder in your Startup menu. You’ll open this folder when you need to access MySQL Workbench. Workbench requires MySQL Server, but you will never use that application directly.

### Downloading ChordCommand Files

This set of steps will guide you through accessing the ***ChordCommand*** installer and loading ***ChordCommand’***s database.

1. Navigate to <https://github.com/charlotte-hirschberg/ChordCommand/releases>
2. You’ll see “Downloads” under the “Latest Release” heading. The “Latest Release” heading will be in large blue font.
3. Don’t worry about downloading source code—you are only interested in two files: ChordCommand-0.1.0.exe and ChordDbDump.sql. Click each of these files in turn to start the download.
4. Again, you’ll see the download start at the bottom of your browser. When the download finishes, click the up arrow beside “ChordCommand-0.1.0.exe” and select “Open.” If you see a “Do you want to allow the following program to make changes to this computer?” message, select “Yes.”
5. The ***ChordCommand*** installer will start. Simply click “Install”, andthe installer will take care of selecting the directory, installing the files, and adding a shortcut to the Start menu. It will try to open ***ChordCommand*** upon completion. You’ll see an error message at that time because the database is not ready. Just close the error dialog, and proceed to the next step.
6. Click the up arrow beside the ChordDbDump.sql download and select “Show in Folder”. Note the location of the file. It will be something like “YourName > Downloads”.

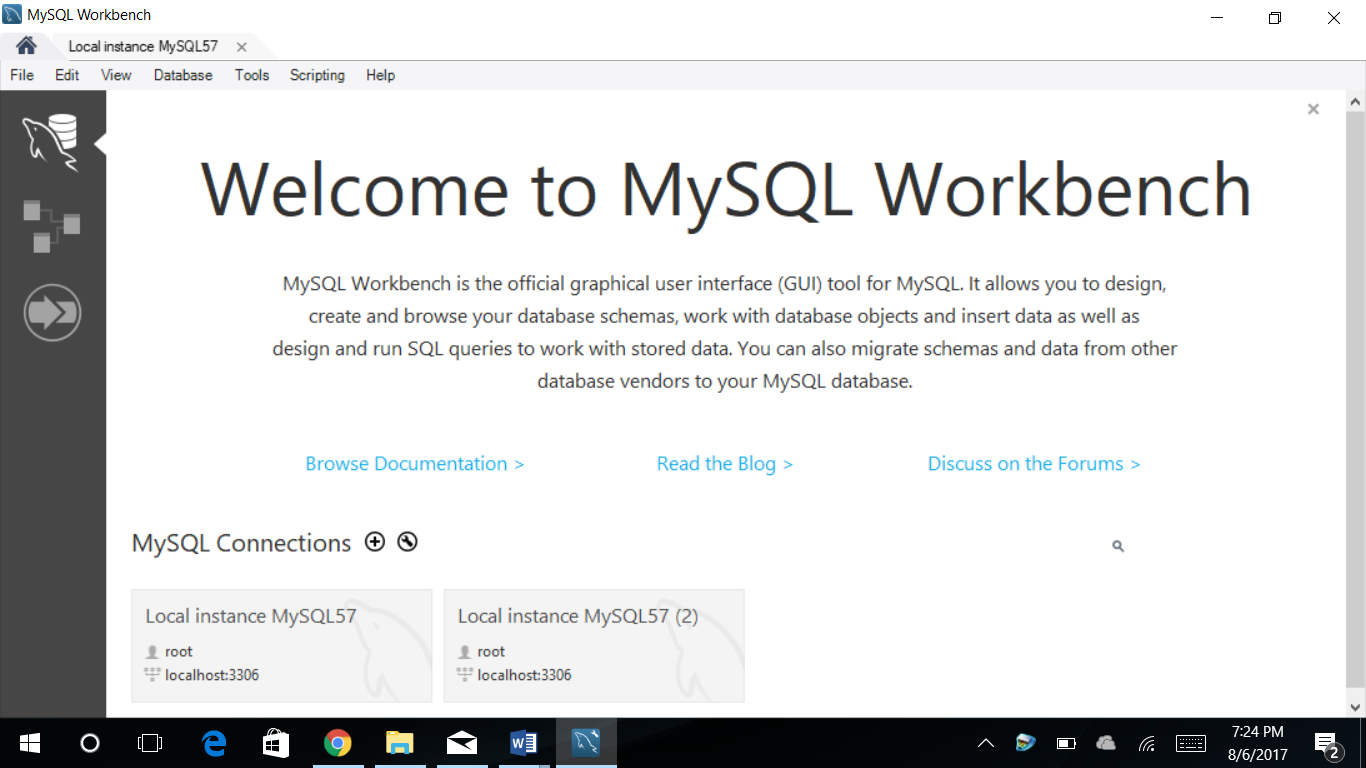
# APPENDIX C (CLIENT SETUP INSTRUCTIONS)

After installing the ***ChordCommand*** files and essential components, you will need to set up MySQL Server, so that it contains data and ***ChordCommand*** can interact with it. You’ll need to perform the following actions:

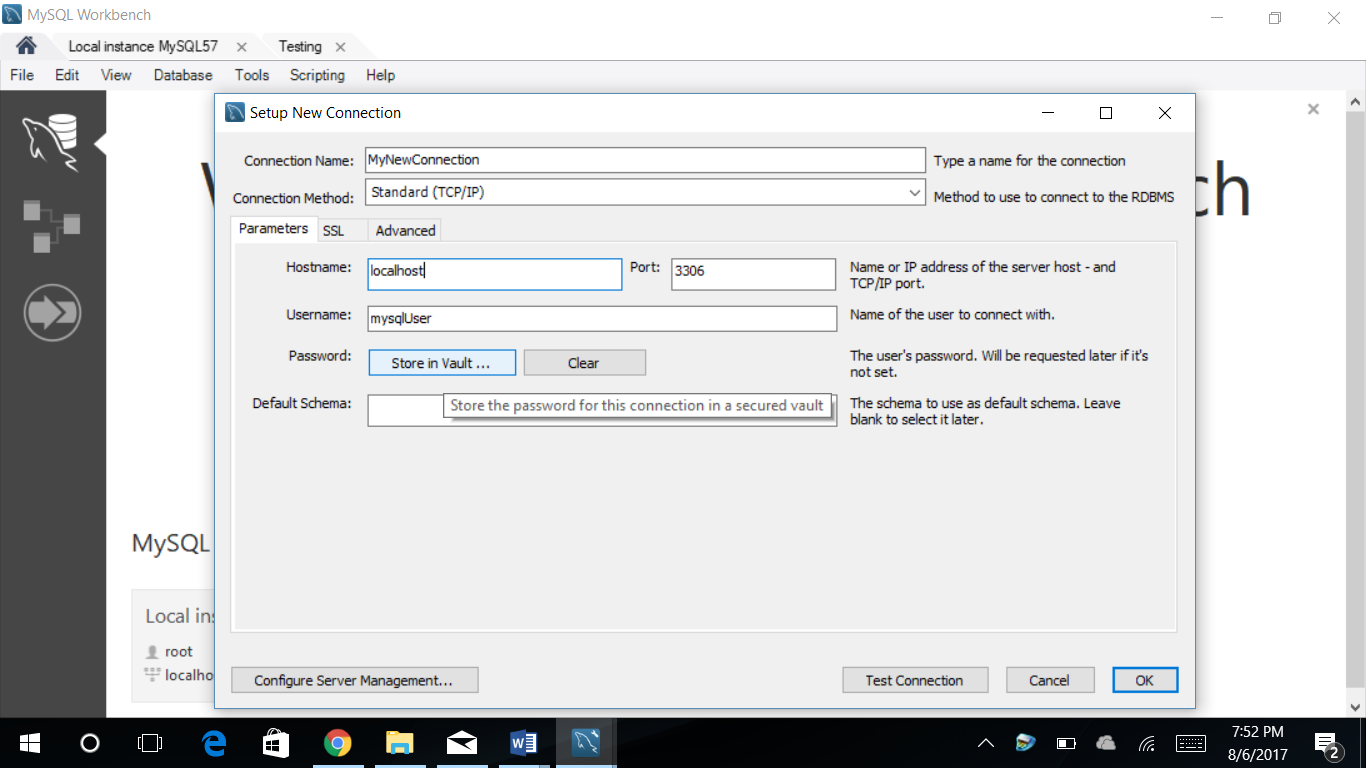
* Configure MySQL Server
* Prepare the Database
* Create the ChordCommand User

### Configure MySQL Server

1. Open MySQL Workbench, using Start > MySQL > MySQL Workbench. From Workbench’s home screen select the + sign next to the heading “MySQL Connections” (outlined in red below).



1. The “Setup New Connection” window appears. Do the following:
   1. Give your connection a name in the “Connection Name” field.
   2. Replace any text in the “Hostname” field with “localhost”.
   3. (Optional) Change the username to whatever you’d like.
   4. Click “Store in Vault” to enter a password. If you don’t enter a password now, you’ll be prompted to select one in the next step.
   5. Leave all the other sections unaltered. The “Port” should be 3306 and the “Connection Method” should be Standard TCP/IP.



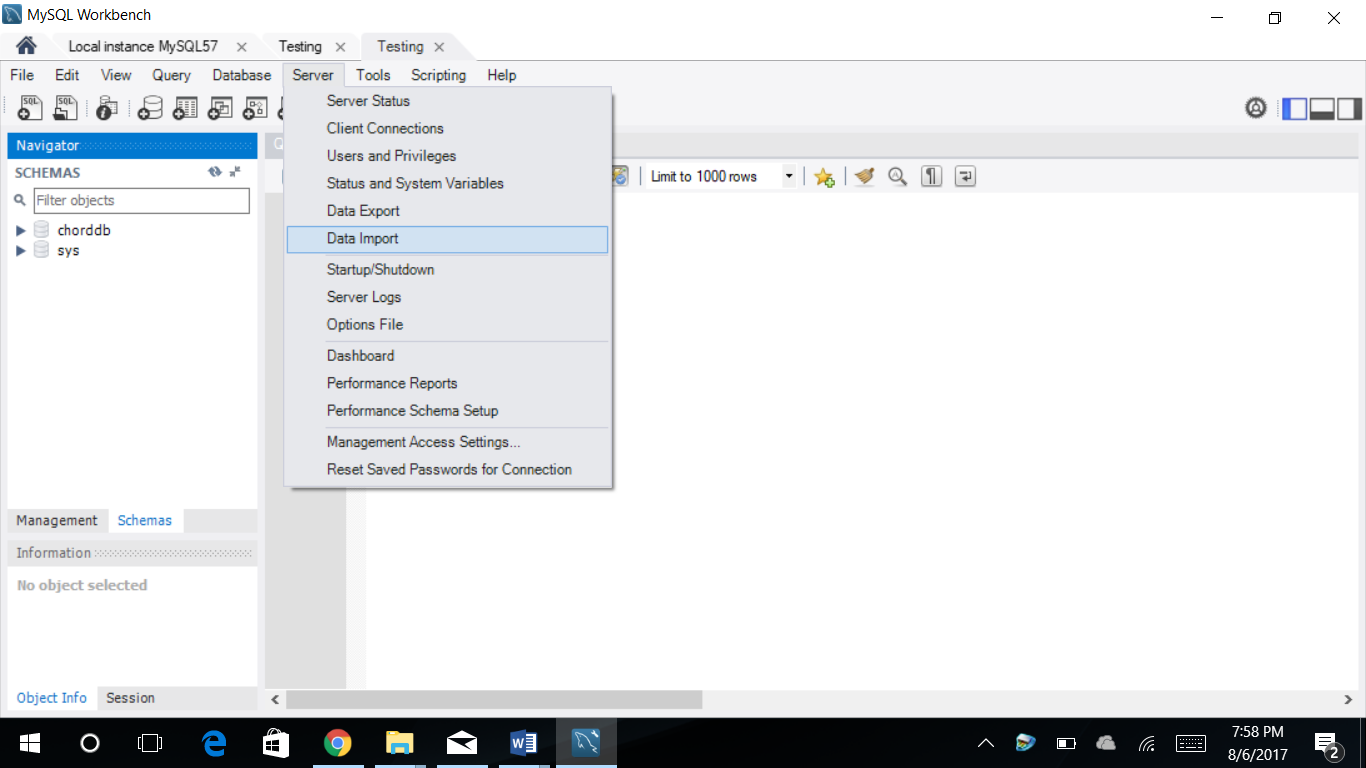
1. Click “OK” to finalize the new connection. It should appear on your home screen.

If you’re having trouble creating a connection, see MySQL’s documentation at: https://dev.mysql.com/doc/workbench/en/wb-getting-started-tutorial-create-connection.html

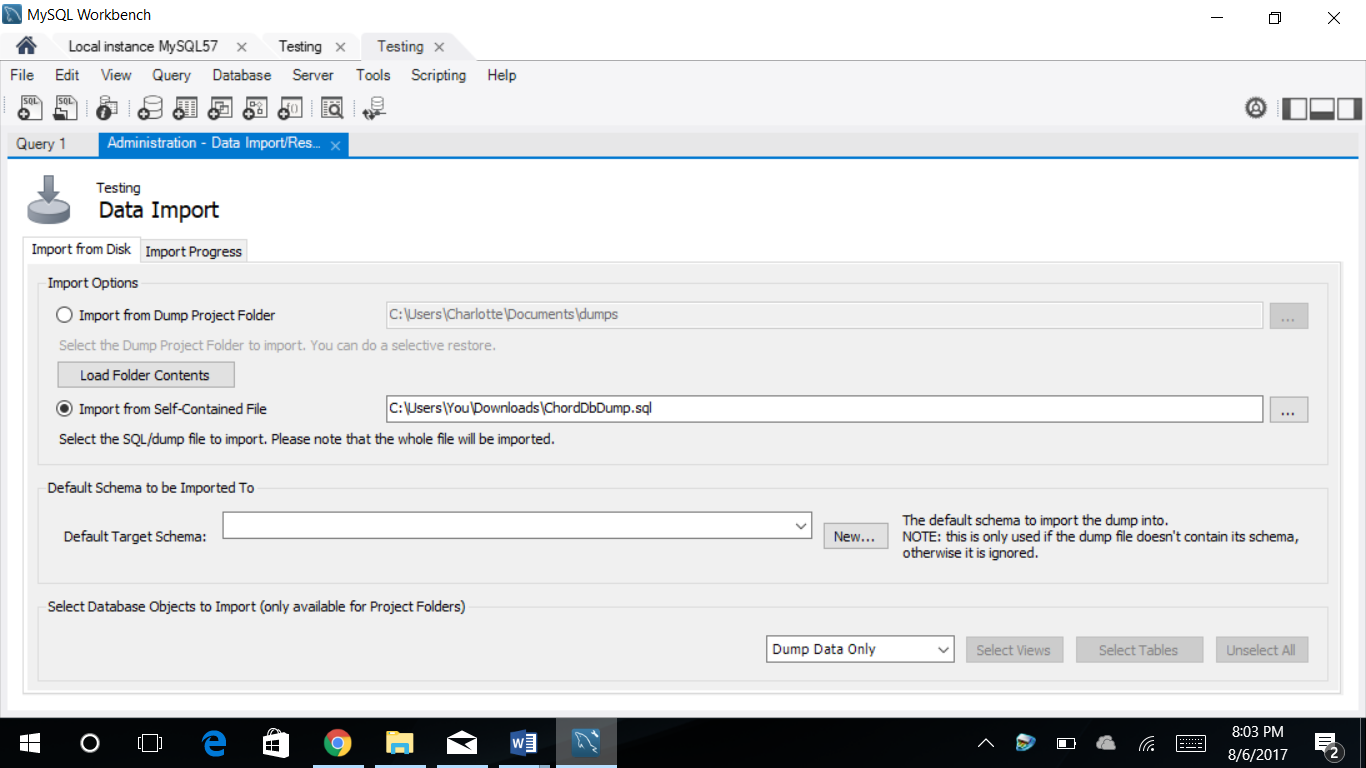
### Prepare the Database

At this point, everything you need has been installed. Follow these steps to add the database tables and data to the MySQL server:

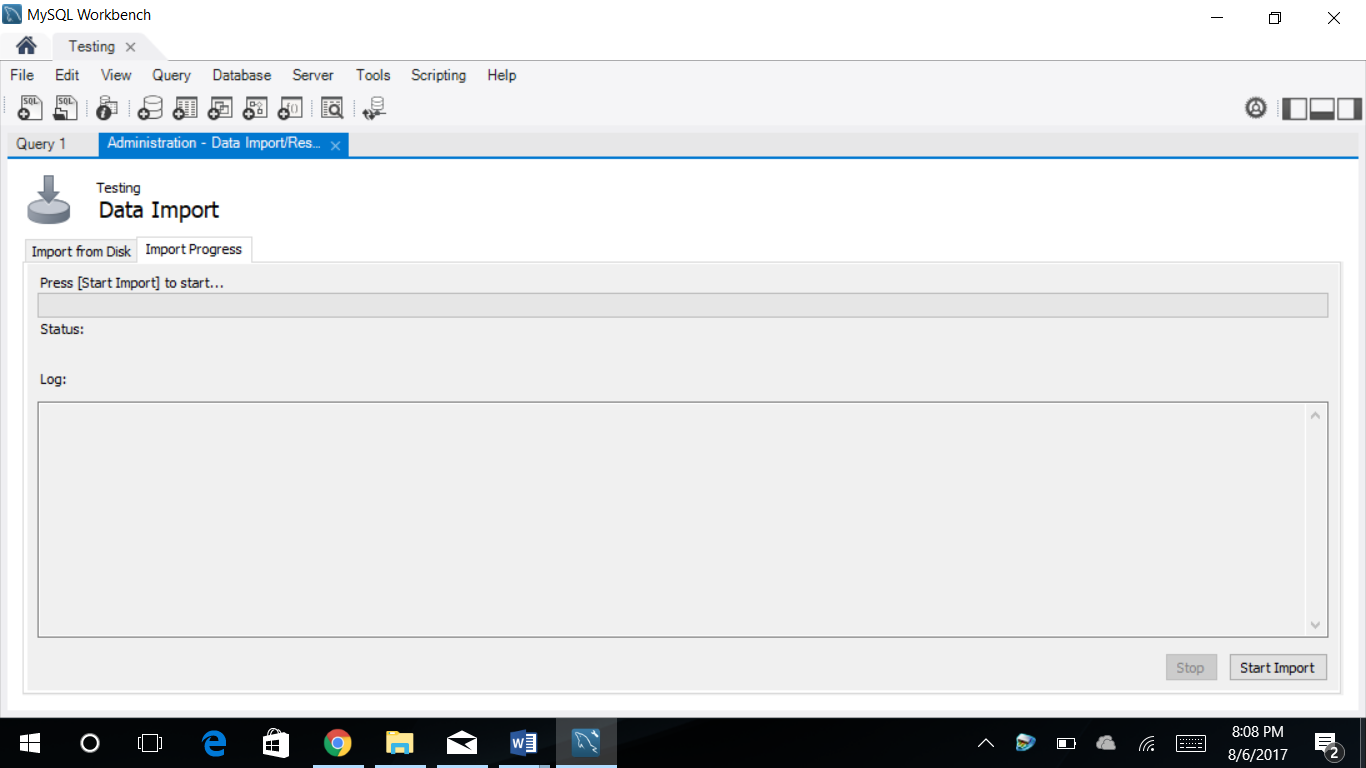
1. Open Workbench if necessary. From the home screen, double click the connection you created earlier.
2. From the screen that appears, select the “Server” menu and then “Data Import.”



1. When the Data Import tab appears, refer to the red outlines in the image below. Make sure the circle next to “Import from Self-Contained File” is selected. Click the button with three dots to navigate to the directory you noted after downloading ChordDbDump.sql. Select ChordDbDump.sql as the file to import. When you’re finished selecting the file, click the “Import Progress” tab (in yellow below).



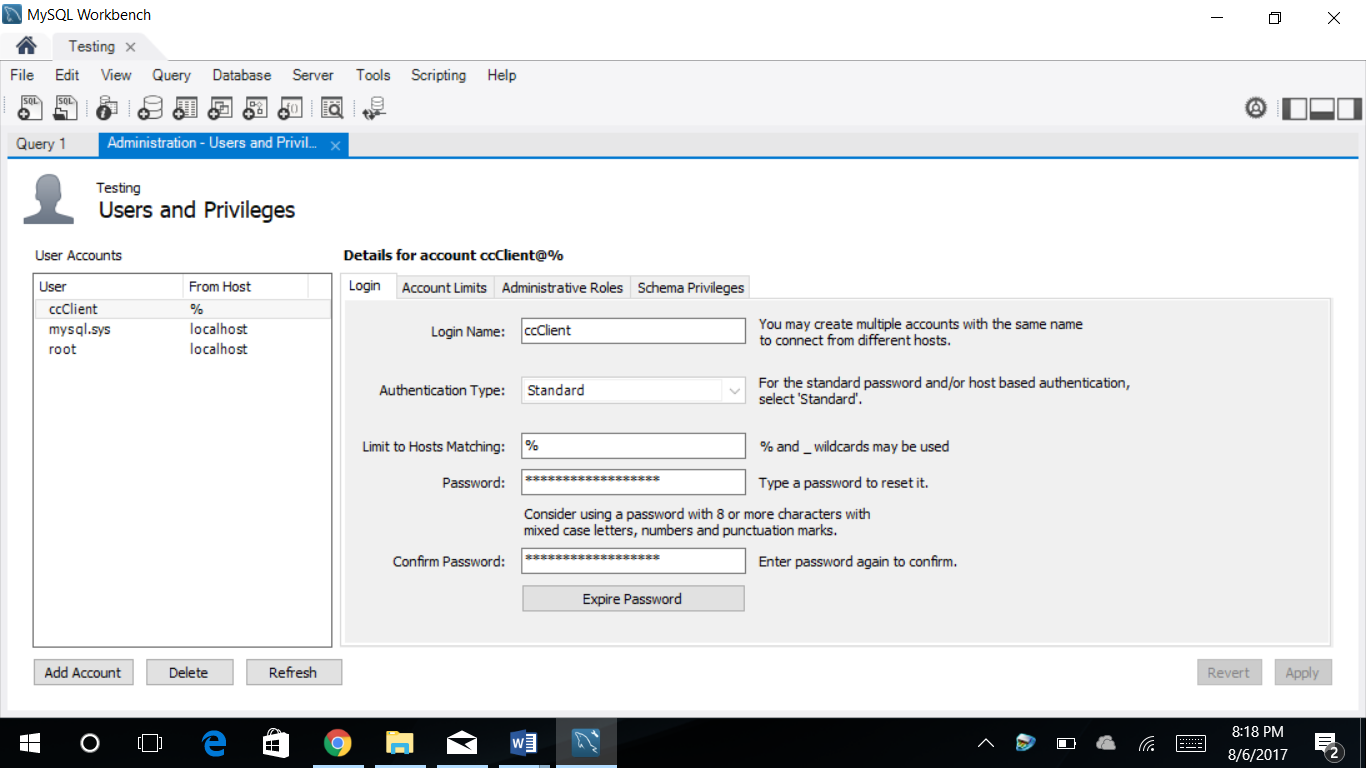
1. From the “Import Progress” tab, click “Start Import” to finish installing the database files.



### Create the ChordCommand User

The final step is to create the User that ***ChordCommand*** functions as while accessing the database:

1. Open the “Server” menu again. Select “Users and Privileges”.
2. Click “Add Account” at the bottom of the screen.



1. In the Login tab, enter the following details:

Login Name: ccClient

Password/Confirm Password: cH0rd$1357

1. Leave the other settings unaltered, and select “Apply” in the bottom right corner. Close Workbench. Congratulations, you should be able to use ***ChordCommand*** now by accessing it from the Start menu!

# APPENDIX D (BUILD AND RELEASE PROCESS)

***ChordCommand*** updates will be released as an .exe that will be available via GitHub release at: <https://github.com/charlotte-hirschberg/ChordCommand/releases>. The latest release will always be tagged “Latest Version.” Developers who want to contribute to ***ChordCommand*** are expected to pull the latest code from the repository and work on their own branch.

Build numbering will follow the Semantic Versioning scheme recommended by GitHub and documented at <http://semver.org/>. The notable features of this scheme include:

* Build numbers take the form Major.Minor.Patch
* Major, Minor, and Patch are all non-negative integers without leading zeroes.
* The Major integer is incremented by 1 when changes are not backwards-compatible
* The Minor integer is incremented by 1 when changes are backwards-compatible and enhance existing functionality
* The Patch integer is incremented by 1 when changes are backwards-compatible and fix a bug
* Major version 0 can be used for initial development, when change is expected to be frequent and updates are likely to conflict with earlier releases
* Hyphens and alphanumeric characters can be used to indicate pre-release versions. According to semver.org, this convention “indicates that the version is unstable and might not satisfy the intended compatibility requirements as denoted by its associated normal version.” For instance, ‘2.0.0-alpha’ could be used to indicate the earliest stages of ***ChordCommand’s*** release of a second version that invalidated code in the first.

As another example, ‘2.2.1’ would indicate that the second major version of ***ChordCommand*** had so far been re-released with 2 enhancements and 1 patch.

At this point, while ***ChordCommand’s*** dominant functionality is in place, it is still likely to undergo frequent changes that are fundamental in nature. Therefore, the team has decided to release the latest .exe as Version 0.1.0 to indicate that it is still in a development-focused state, rather than a stable, production-ready one. Until ***ChordCommand*** is deemed ready for its first major release, increments of Minor and Patch will be used to indicate the nature of an update (enhancement or bug), regardless of compatibility.

# APPENDIX E (TESTING PLAN)

***ChordCommand’s*** testing plan was devised to focus on the processing of chord entries and the interaction between user preferences and output. These areas were the focus of testing for three reasons:

1. They are non-standard programming tasks.
2. They are critical to the achievement of ***ChordCommand’s*** fundamental objectives.
3. They require a complex application of logic to combinations of input.

Decision-table and task-oriented scenario models were included. Decision-table testing was used for preferences-related tasks, owing to the multitude of input combinations inherent to those tasks. A more traditional approach was used for the remaining functionality; that approach relied on isolating and assessing individual tasks that a user is likely to perform.

As shown in the documents that follow, corrective action was taken and recorded for any test case that didn’t initially succeed. While not exhaustive, test cases were augmented by a quality-driven approach and ad-hoc testing throughout the project. These test cases verify the functionality of ***ChordCommand’s*** core features.

### Testing Output

These tests are designed to verify the interaction between preferences and output, while simultaneously (but cursorily) testing validation. The textfield input that produces failed tests should be noted.

False = preference set to false

True = preference set to true

X = passed

! = failed

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Cases** | TC1 | TC2 | TC3 | TC4 | TC5 | TC6 | TC7 |
| **Conditions** |  |  |  |  |  |  |  |
| Invalid Chord Entry | True | True | False | False | False | False | False |
| Chord Not in DB | -- | -- | True | False | False | False | False |
| Show Chord Numbers | False | False | False | True | False | True | False |
| Show Piano | False | False | False | False | True | True | False |
| Show Scales | False | True | True | False | True | False | False |
| Show Scale Numbers | False | False | False | False | True | False | True |
| Show WH Form | False | False | False | False | False | True | True |
|  |  |  |  |  |  |  |  |
| **Output** |  |  |  |  |  |  |  |
| Alert invalid entry | X | X |  |  |  |  |  |
| Alert not found |  |  | X |  |  |  |  |
| Show chord pitches |  |  |  | X | X | X | X |
| Show chord numbers |  |  |  | X |  | X |  |
| Show piano output |  |  |  |  | X | X |  |
| Show scales window |  | X | X |  | X |  |  |
| Show scale numbers |  |  |  |  | X |  |  |
| Show WH forms |  |  |  |  |  |  |  |

Percent Passed: 100%

Percent Failed: 0%

Comments: Consider whether existing output should be erased when an invalid/unfound entry is made. A preference to disable piano/scale views should be reflected in the output when Preferences are submitted, not when an entry is.

### Testing Storage of Preferences

Start these tests with all properties set to false. In addition to testing retrieving and persisting preferences, these tests are designed to verify the interaction between the Show Scales checkbox and its children.

False = checkbox not selected

True = checkbox selected

X = passed

! = failed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Cases** | TC1 | TC2 | TC3 | TC4 | TC5 | TC6 |
| **Conditions** |  |  |  |  |  |  |
| Show Chord Numbers | False | False | True | True | False | True |
| Show Piano | False | False | True | True | False | True |
| Show Scales | False | True | True | True | False | False |
| Show Scale Numbers | False | False | False | True | True | False |
| Show WH Form | False | False | False | False | True | True |
|  |  |  |  |  |  |  |
| **Output** |  |  |  |  |  |  |
| All properties set to false | X |  |  |  |  |  |
| Chord properties set to true |  |  | X | X |  | X |
| ShowAnyScale set to true |  | X |  | X |  |  |
| ShowScaleNum set to true |  |  |  | X | ! |  |
| ShowWH set to true |  |  |  |  | ! | ! |
| Show Scales selected on next load |  | X | X | X |  |  |
| Scales>>Numeric selected on next load |  |  |  | X | ! |  |
| Scales>>WH Form selected on next load |  |  |  |  | ! | ! |
| Scales>>Numeric disabled on next load | X |  |  |  | X | X |
| Scales>>WH Form disabled on next load | X |  |  |  | X | X |

Percent Passed: 17/23 = 74%

Percent Failed: 6/23 = 26%

Comments: Repeat TC5 and TC6 after fixing program logic. Fixed on 8/3/17.

### Retesting Cases TC5 and TC6

|  |  |  |
| --- | --- | --- |
| **Test Cases** | TC5 | TC6 |
| **Conditions** |  |  |
| Show Chord Numbers | False | True |
| Show Piano | False | True |
| Show Scales | False | False |
| Show Scale Numbers | True | False |
| Show WH Form | True | True |
|  |  |  |
| **Output** |  |  |
| All properties set to false |  |  |
| Chord properties set to true |  | X |
| ShowAnyScale set to true |  |  |
| ShowScaleNum set to true | X |  |
| ShowWH set to true | X | X |
| Show Scales selected on next load |  |  |
| Scales>>Numeric selected on next load | X |  |
| Scales>>WH Form selected on next load | X | X |
| Scales>>Numeric disabled on next load | X | X |
| Scales>>WH Form disabled on next load | X | X |

Percent Passed: 100%

Percent Failed: 0%

Comments: Repeated TC5 and TC6 after fixing program logic.

(cont. on next page)

### Testing Checksum Methods

These cases test the interaction between the main method and the ChecksumUtil class on startup and exit. These interactions are responsible for verifying the integrity of files and taking corrective action when they are found to be corrupt.

F = condition does not apply

True = condition applies

‘-‘ = non-applicable

X = passed

! = failed

**Database Properties file**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Cases** | TC1 | TC2 | TC3 | TC4 | TC5 | TC6 | TC7 | TC8 | TC9 | TC10 | TC11 | TC12 |
| **Conditions** |  |  |  |  |  |  |  |  |  |  |  |  |
| DB algorithm exception | T |  | T | T | F | F | F | F | F | F | F | F |
| DB check file exception | - |  | - | - | T | T | T | F | F | F | F | F |
| DB check IO exception | - |  | - | - | - | - | - | F | F | F | F | F |
| Proceed? | T |  | T | F | T | T | F | - | - | T | T | F |
| Checksums match? | - |  | - | - | - | - | - | T | T | F | F | F |
| DB properties loaded? | T |  | F | - | T | F | - | T | F | T | F | - |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Output** |  |  |  |  |  |  |  |  |  |  |  |  |
| Update Java alert | x |  | x | x |  |  |  |  |  |  |  |  |
| Connection error alert |  |  | x |  |  | x |  |  | x |  | x |  |
| ‘Will try to fix’ alert |  |  |  |  | x | x | x |  |  |  |  |  |
| On exit, create .chk |  |  |  |  | x |  |  |  |  | x |  |  |
| Exit immediately |  |  | x | x |  | x | x |  | x |  | x | x |
| Suggest restarting |  |  |  |  |  |  |  |  |  |  |  |  |
| Alert mismatch |  |  |  |  |  |  |  |  |  | x | x | x |
| No visible effect |  |  |  |  |  |  |  | x |  |  |  |  |

Percent Passed: 100%

Percent Failed: 0%

### Testing Checksum Methods, cont.

**Preferences Properties File**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Algorithm exception | T |  | F | F | F | F |
| File exception | - |  | T | F | F | F |
| Checksums match? | - |  | - | T | T | F |
| Preferences loaded? | - |  | - | T | F | - |
| Use defaults? | T |  | T | - | - | - |
|  |  |  |  |  |  |  |
| Update Java alert | x |  |  |  |  |  |
| Load error alert |  |  |  |  | x |  |
| ‘Can try to fix’ alert |  |  | x |  |  |  |
| On exit, create .chk |  |  | x | x |  | x |
| Suggest restarting |  |  |  |  |  |  |
| Load defaults | x |  | x |  | x | x |
| On exit, store |  |  | x | x | x | x |
| On exit, compare |  |  |  |  |  |  |
| Alert mismatch |  |  |  |  |  | x |

Percent Passed: 100%

Percent Failed: 0%

### Overview – Task-Based Testing



### Help & Startup Testing



### Root Layout Testing



### Preferences Testing



### ChordView Testing

