

COE848 Lab 5  
Daniel Medetbayev  
500890241

MedusaUI Documentation  
V 1.03

## Table of Contents

---

### 1. Introduction

- 1.1. Version overview statement
- 1.2. Version features overview
- 1.3. Version disclaimers
- 1.4. Database flow
- 1.5. UI Mockup
- 1.6. Requirements to run

### 2. Functionalities

- 2.1. General UI overview
- 2.2. Create session rapid flow
- 2.3. Create session complete flow
- 2.4. Open session flow
- 2.5. View profile base flow
- 2.6. Modify profile flow
- 2.7. Light profile creation flow
- 2.8. Statistics flow

### 3. Current issues/limitations

### 4. Appendix

## 1. Introduction

---

*"V2.04 features a new analytics feature for users, where they can view various statistics regarding their profile's histories including the chat and change logs. Additional data provided is in the form of word with the most tags, the last successfully closed session, and the longest chat session for a profile. A small change was also made to the config section of profiles, where the data is now compared against the previously entered value for a modification. Unfortunately, no data saving was completed, so all of the changes are still only local!"*

---

### 1.1

Version 2.04 of the Medusa UI Project features more up-to-date functionality and a full 10 queries that help interact with the preset data. There were a few major factors that led to many of the previous objectives from V1.03 not being completed, however for the purposes of demonstrating database connectivity and the integration of queries into a java program, V2.04 fully meets the requirements.

---

### 1.2

The new list of functionalities includes full reading and searching from the preset database that comes with the program, as well as an updated list of queries utilised for obtaining many of the analytics-based data.

The new statistics and data logs page features all data from the chat logs and change logs recorded as part of the sample preset database, as well as some bonus data that inform the user of the various information collected on their interactions with a preset.

---

### 1.3

Testing was done via the NetBeans IDE and so that is the recommended tool for running the program. Tests on machines foreign to the development machine were used to ensure integration testing as well as used to root out potential errors in the program, and from the resulting data no major issues were found.

Some functionality in the program is fairly unintuitive and at times the lack of immediate program response may create the feeling that the program either broke or is performing in an unexpected way, however this behaviour was kept as-is for V2.04 due to time limitations. Portions of note that display this behaviour are the *Modify Profile*, *Modify Configuration*, and *Fork Profile* screens.

This being said, the potential for an error of some form is always present and cannot be guaranteed to have been detected within the time taken to develop and test the program. If any errors occur, it would be appreciated that it be noted of to assist with the creation of a better experience in the full version of the program.

---

### 1.4

The MedusaUI program was made in mind with the question of how data should flow between entities, and that design choice can be seen in all elements across the program. Tight clustering of entities which frequently communicate with each other was the key to the format and resulted in the flow diagram *Figure A1* found in the appendix. Seeing as this was the basis for the whole program, a lot of the desire was to fixate the ER Diagram (*Figure A2* in the appendix) in the same manner, and the clustering was especially important to determine where an entity should be in relation to another.

The main method to try and remove the circular nature of the ER Diagram was to restrict "2-way" communication patterns between entities. If an entity-relation can be described in a form where entity A can write to entity B and vice-versa, then that would be an invalid format and a relation between entities cannot be properly formed. This approach led to thinking of the program implementation early on as well as early design of the structure of the tables.

---

## 1.5

The UI mockup for the program was designed in Balsamiq Mockups 3 and was used to plan the layout and flows of the MedusaUI program. As seen under *Figure A3* in the appendix, the mockup shows the various screens in different flows associated with the program. The design is not consistent with V2.04 as some features have been left out due to time constraints such as the deep configuration flow or the alerts shown in the delete entry and fork profile screens. A splash loading screen for the program bootup was also not included in version 2.04 due to time constraints.

---

## 1.6

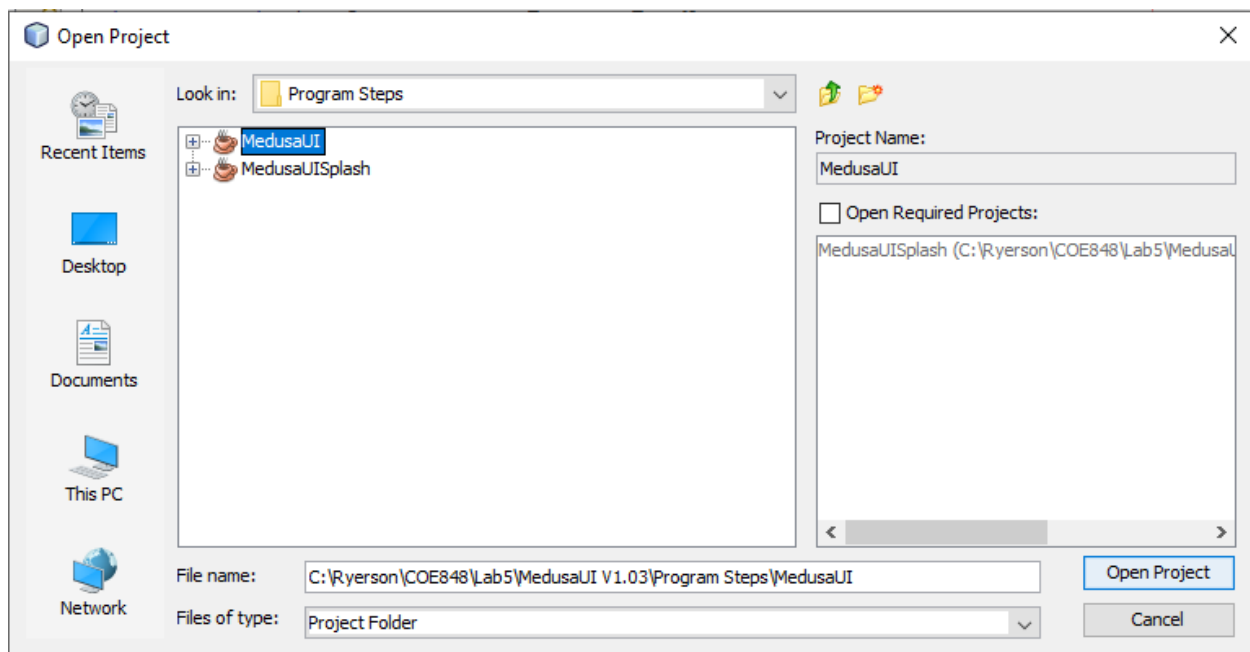
As mentioned before, NetBeans was used in order to develop and test the program, and the files itself will be provided in the form of the original netbeans project files along with all the additional files that were included inside.

Additionally, the program DB Browser (SQLite) was used in order to create and make changes to the database as well as test and format queries used in V2.04 of the program. It may not be necessary to create and populate the database, however for the

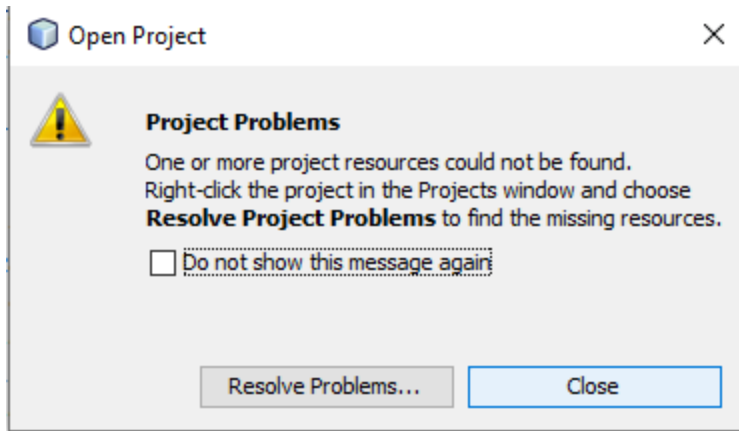
purposes of this guide the steps involving the database portion will be including it. The link to the software is provided here: <https://sqlitebrowser.org/dl/>

The project file should have the filled database as part of the program files, however if you wish to install the database from scratch, use the procedure listed below. Do note, however, the "MedusaUI.db" file located in the "MedusaUI" folder needs to be deleted before progressing with the steps outlined below.

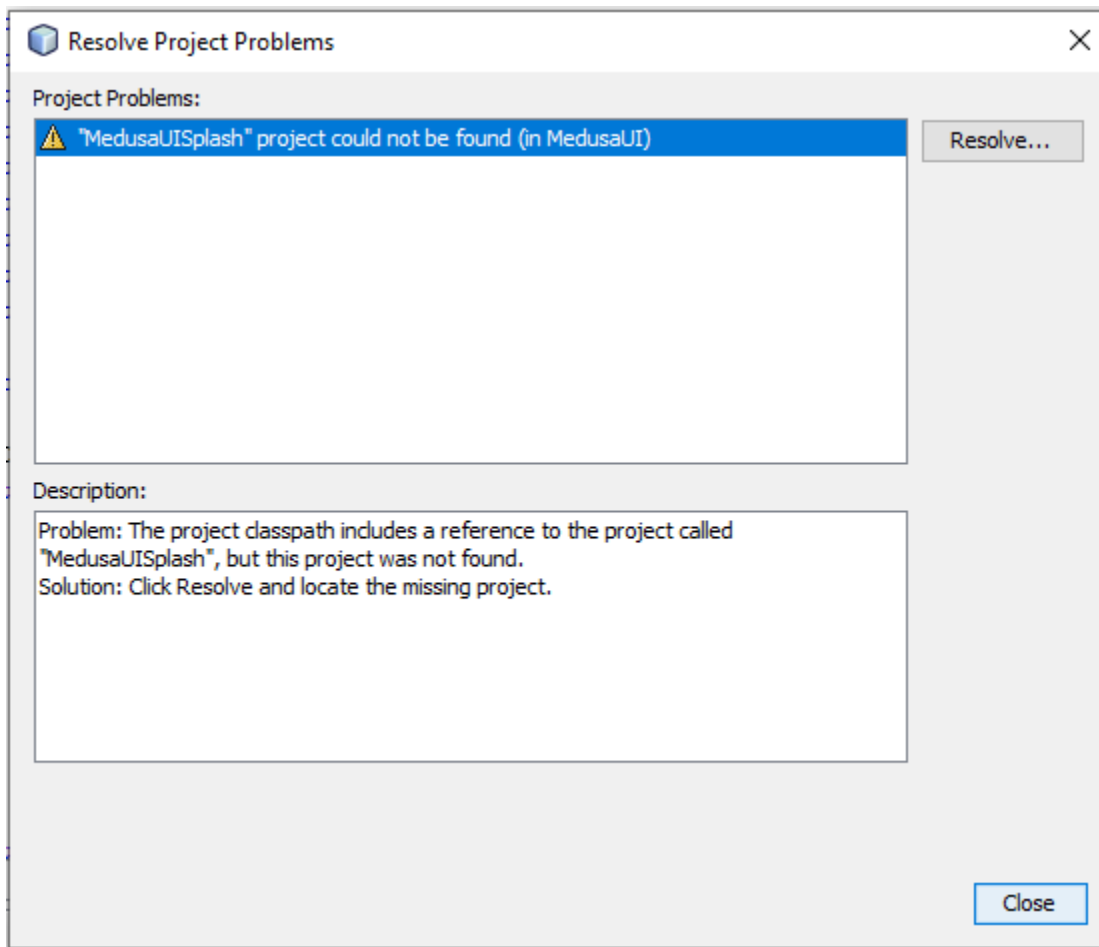
For the program, Netbeans is used. Using the installer found in the project folder, simply run it and click on next through all of the options. When inside the actual IDE, click on "File" in the top left corner and click on "Open Project".



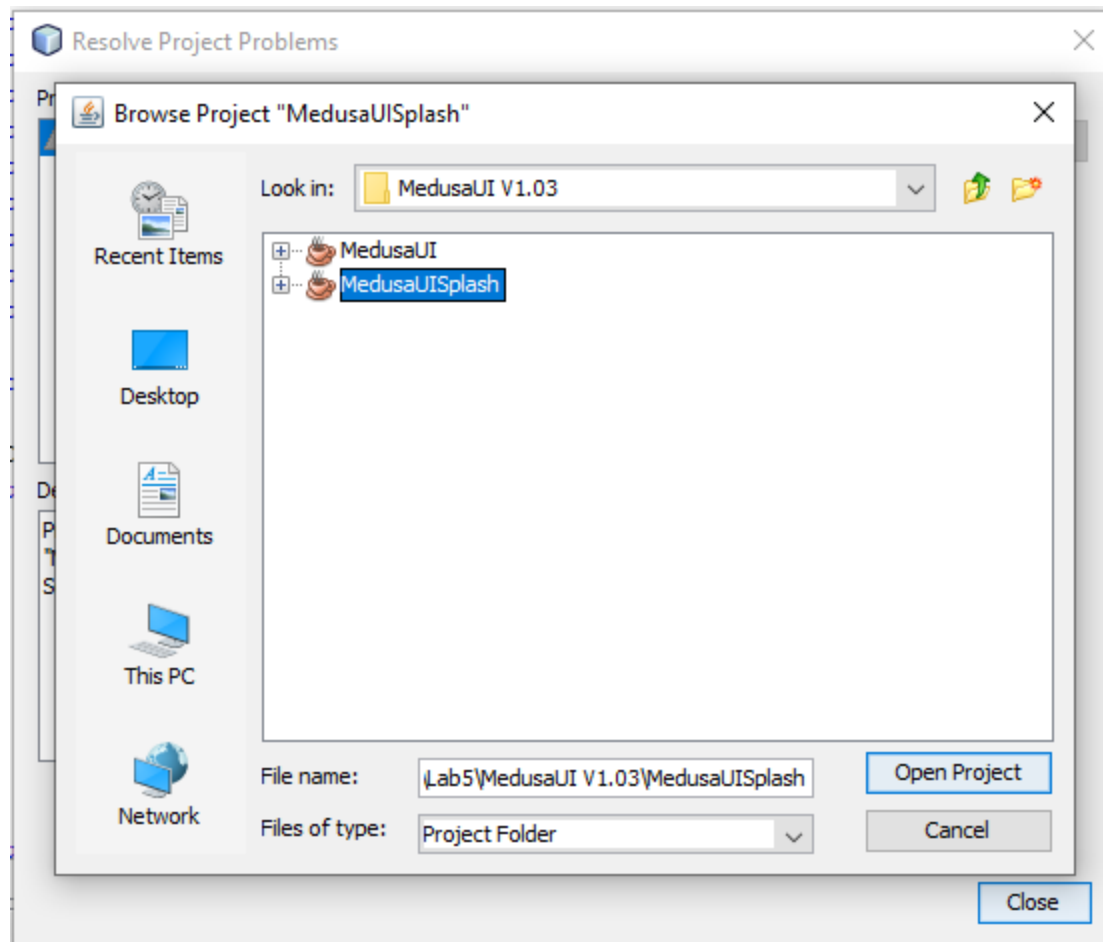
A resolve conflict screen should now appear. Click on "Resolve Problems".



From here click on "Resolve"

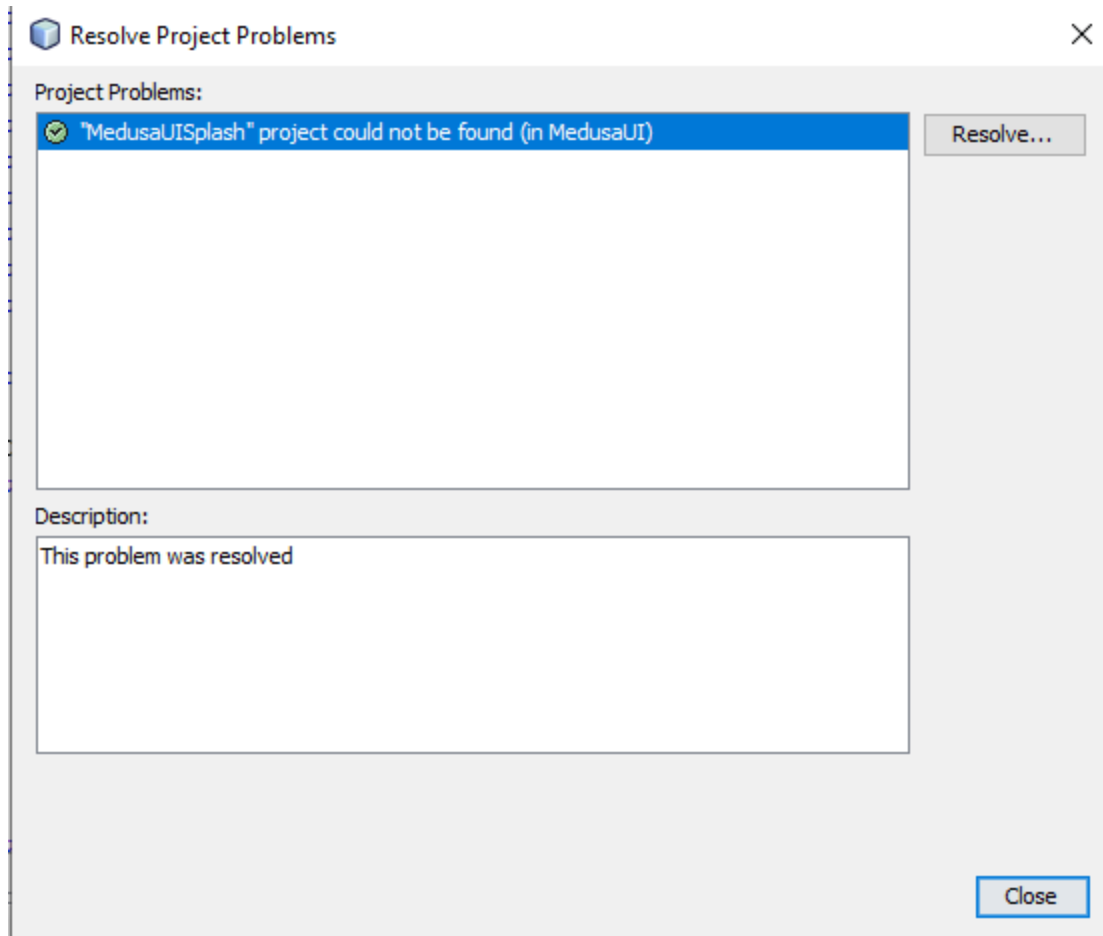


Press on the MedusaUISplash project file that is found in the MedusaUI V2.04 folder, and press "Open Project"



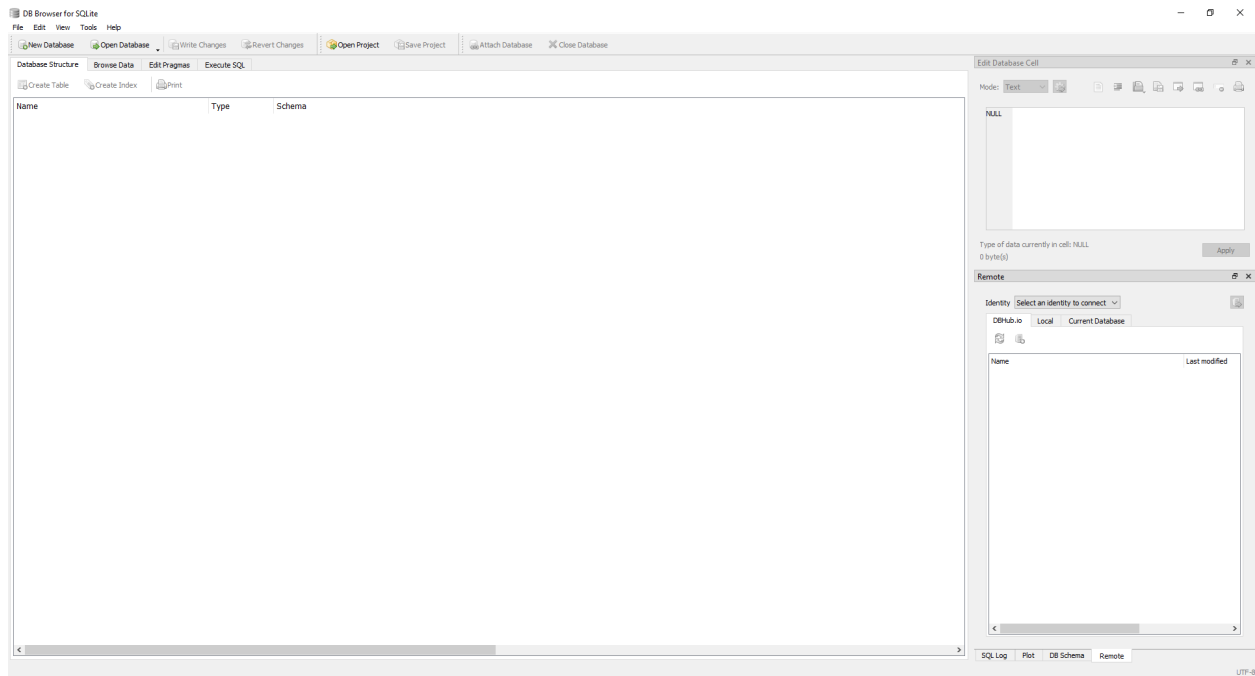
From here the conflict should be resolved, and you should be able to open and run the project.



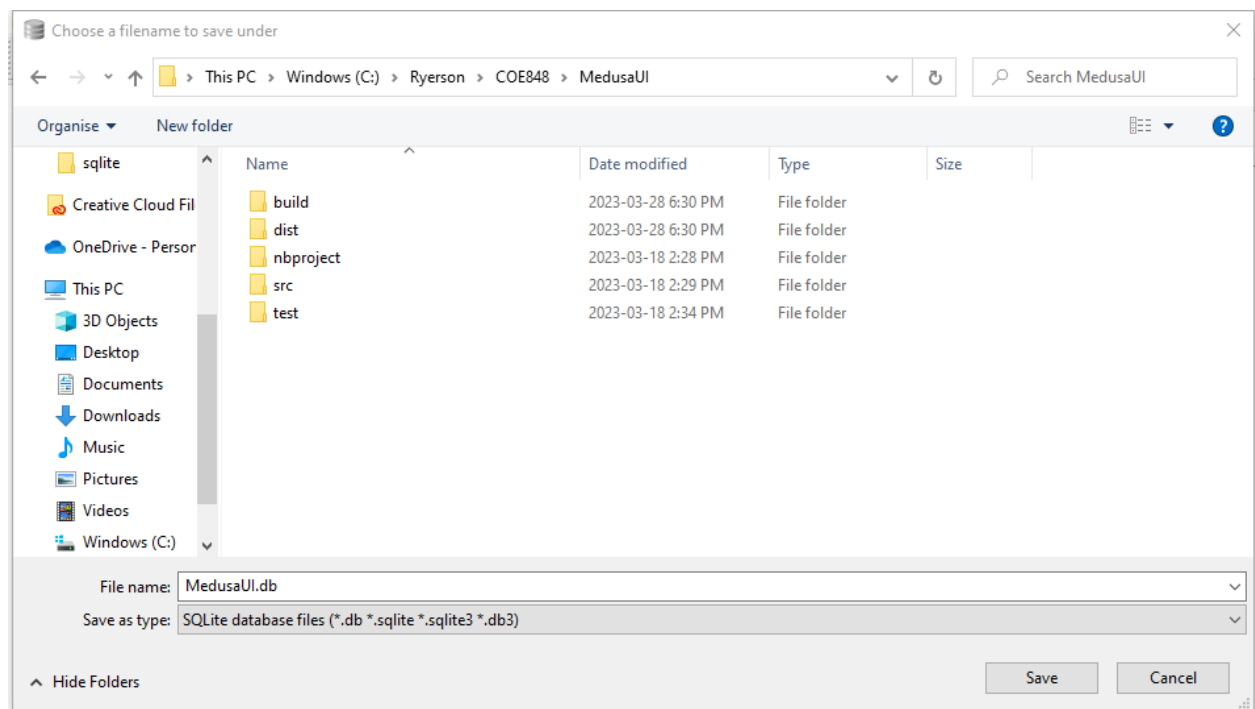


That's it! The project file should already have all of the required libraries included in the file. Even though the MedusaUISplash is not used, it still needs to be kept around as part of the project's configuration.

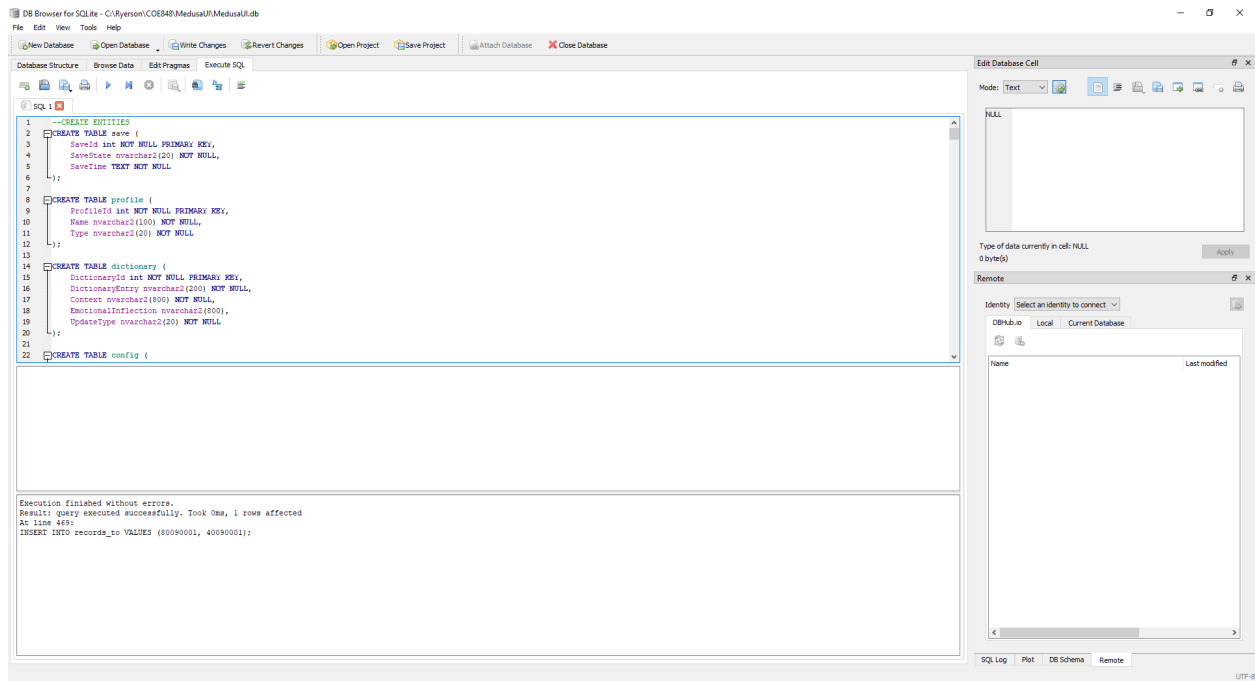
For the database, a file with the updated database structure and the commands to create and populate all the tables is attached. Opening DB Browser, the screen should look as in the screenshot below. Press the "New Database" button found in the top left.



Once the menu opens up, navigate to the project folder of the MedusaUI application and name the new database as "MedusaUI.db". The program is sensitive with respect to the database, so it is required that the name and location is accurate.



Once created, navigate to the "Execute SQL" tab in the program, and paste the contents of the "MUIDBStructure.txt" file in the textbox, and click the blue triangle to execute the commands.



Once completed, be sure to click "Write Changes" at the top of the screen in order to commit changes to the database. From here, the program is ready for use.

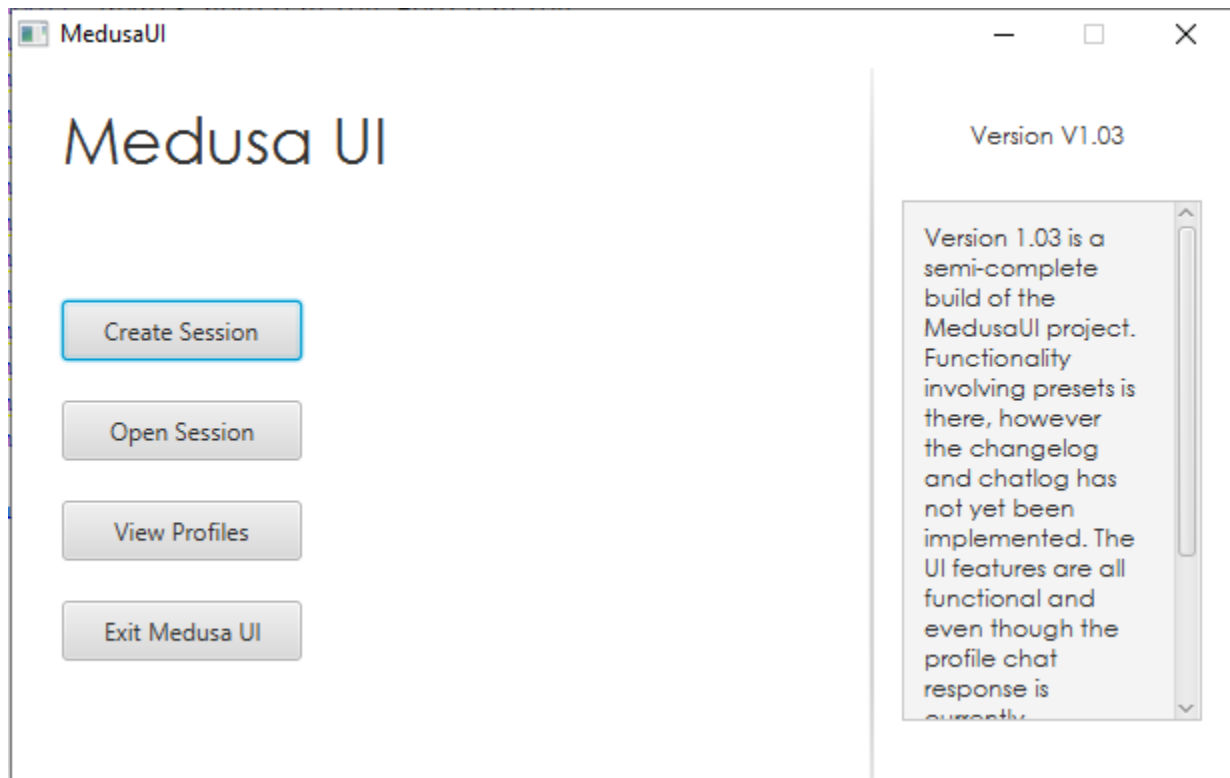
## 2.1

The overall UI features many basic user IO functionality that is designed to facilitate user intractability with the program's features. Most of what is used involves visual grids of information that are populated once the desired selections are made. Most of what is implemented is functional at a local level with exception of retrieving, which is fully implemented with the database.

Queries that are currently implemented as a part of the current version will be highlighted where they appear as parts of the flow overviews, but additionally within the appendix as

respective figures. What is worth noting is that in each flow overview there will be a sample sequence of prompts to enter that can be repeated to guarantee that the program functions for those specified prompts.

All of the program starts from the homescreen, which features a scroll pane of the version overview as well as the options that a user can take on.

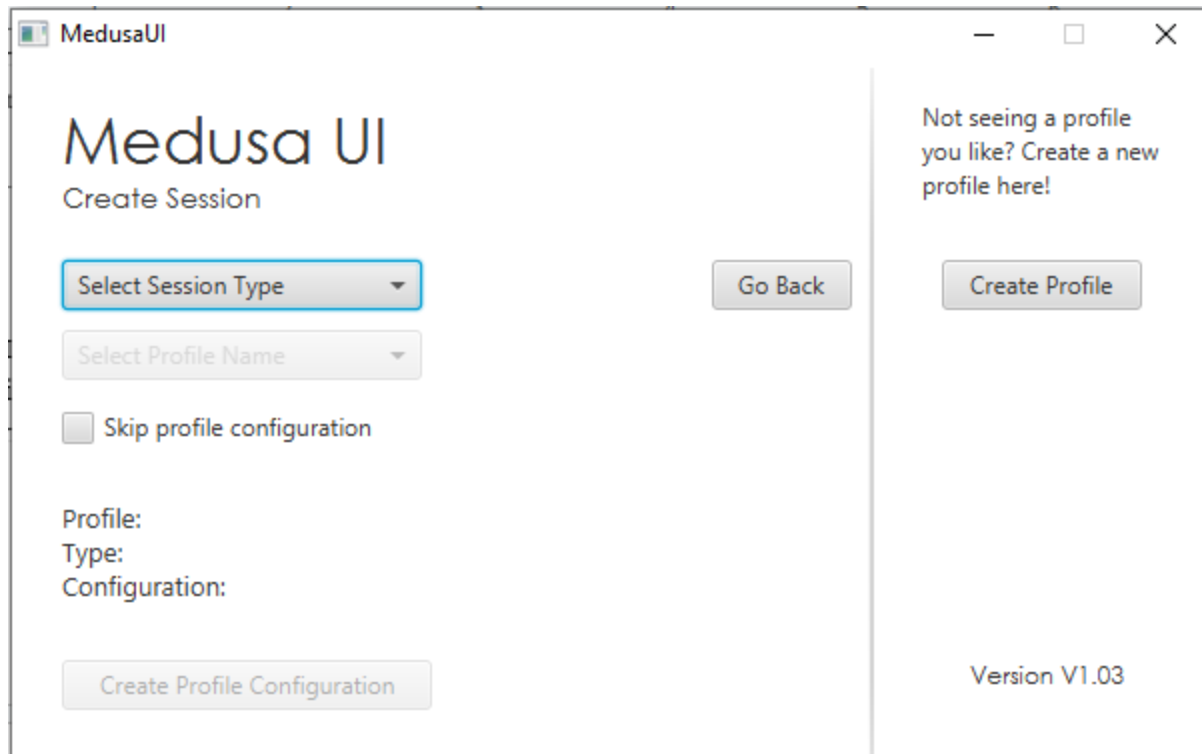


The exit option simply closes the program, while the remaining 3 options are all the first steps to engaging many of the implemented flows.

---

## 2.2

In the create session rapid flow, the first prompt for a user is to click the "Create Session" button. The respective screen would then be loaded.



On this screen there are a multitude of options, including "Go Back", "Create Profile", and then the main body of the create session screen in the form of the 2 comboboxes, a checkbox, and a button. Options in all flows are disabled until a previous step of the flow is successfully inputted and verified. A session type must be selected to progress. The MedusaUI program allows 2 types : "PRESET" and "CUSTOM" profiles. A "PRESET" profile is one that exists as part of the program on launch, and is configured with a unique configuration and dictionary set. A custom profile is a user created profile that can be customised in many different ways. Since no custom users exist on launch, select the "PRESET" option.

MedusaUI

# Medusa UI

Create Session

PRESET

Select Profile Name

☐ Skip profile configuration

Profile:  
Type: PRESET  
Configuration: PRESET configuration

Create Profile Configuration

Go Back

Create Profile

Not seeing a profile you like? Create a new profile here!

Version V1.03

Following the type selection, the "Select Profile Name" combobox is now enabled. As part of the sample input, select the "Athena" option.

# Medusa UI

Create Session

PRESET ▼

Go Back

Create Profile

Athena ▼

☐ Skip profile configuration

Profile: Athena

Type: PRESET

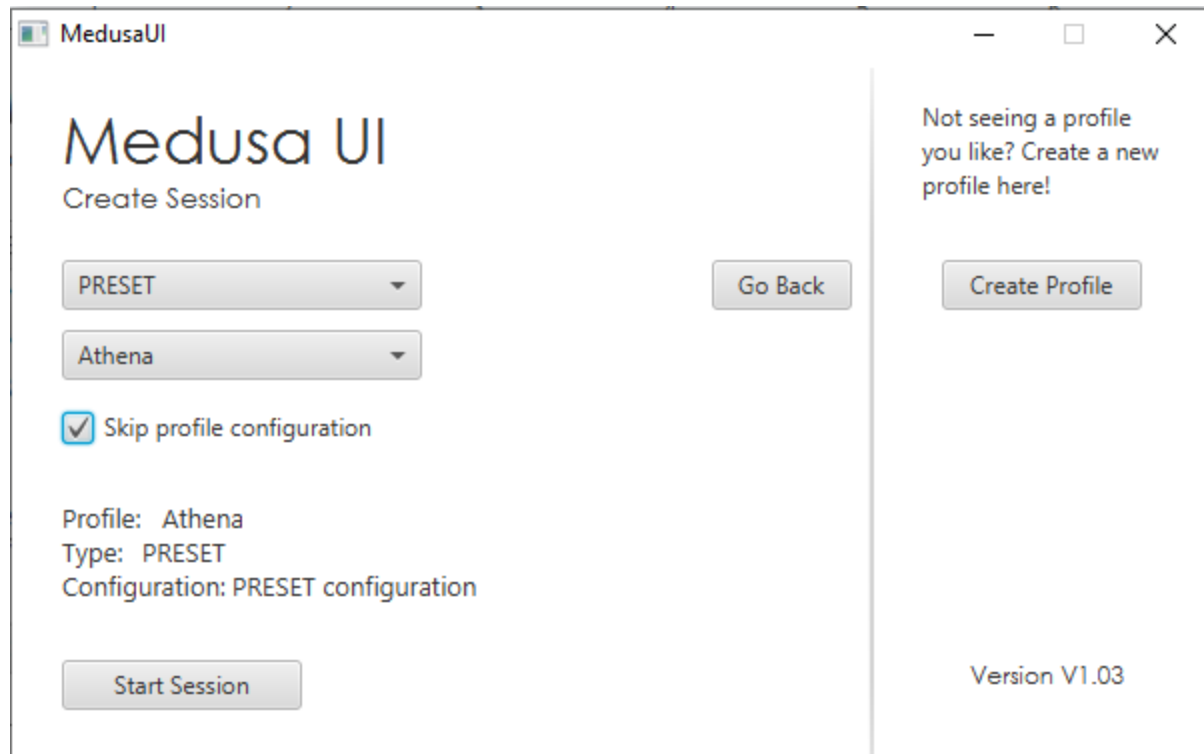
Configuration: PRESET configuration

Create Profile Configuration

Not seeing a profile  
you like? Create a new  
profile here!

Version V1.03

To enter the rapid flow, select the "Skip profile configuration" checkbox. The "Create Profile Configuration" button should be set to "Start Session".



The image shows a software window titled "MedusaUI" with standard Windows window controls (minimize, maximize, close). The window is divided into two main sections. The left section, titled "Medusa UI" and "Create Session", contains two dropdown menus: the first is set to "PRESET" and the second to "Athena". Below these is a checked checkbox labeled "Skip profile configuration". Further down, it displays the selected values: "Profile: Athena", "Type: PRESET", and "Configuration: PRESET configuration". At the bottom of this section is a "Start Session" button. The right section contains a message: "Not seeing a profile you like? Create a new profile here!" with a "Create Profile" button. A "Go Back" button is also present between the two sections. At the bottom right of the window, the text "Version V1.03" is displayed.

Medusa UI

Create Session

PRESET

Athena

☒ Skip profile configuration

Profile: Athena  
Type: PRESET  
Configuration: PRESET configuration

Start Session

Go Back

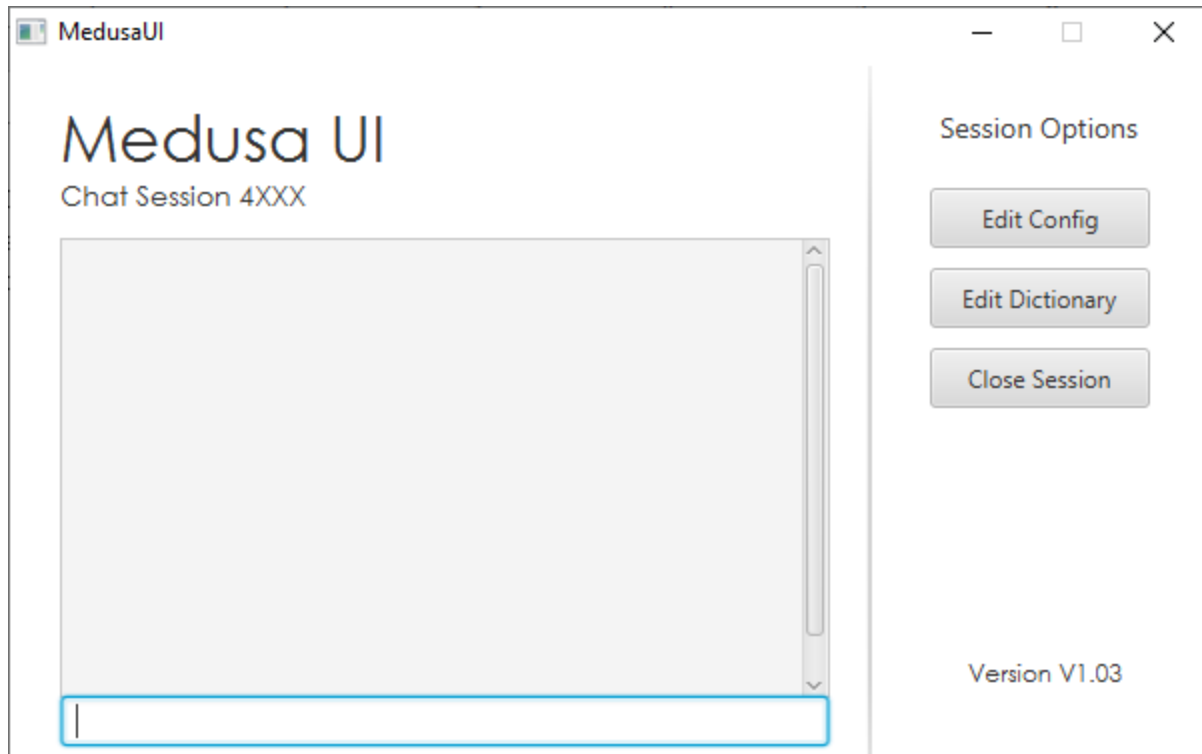
Create Profile

Not seeing a profile you like? Create a new profile here!

Version V1.03

The user should now be at the "Chat Session" screen. This is the end of the create session rapid flow. Click the "Close Session" button to return to the home screen.





---

### 2.3

The create session complete flow also starts at the "Create Session" screen, however the options are slightly different. Repeat the previous steps until the checkbox option. Here, the checkbox will be kept unchecked, and instead the user should press the "Create Profile Configuration" button.

Medusa UI

Create Session

PRESET

Go Back

Create Profile

Athena

☐ Skip profile configuration

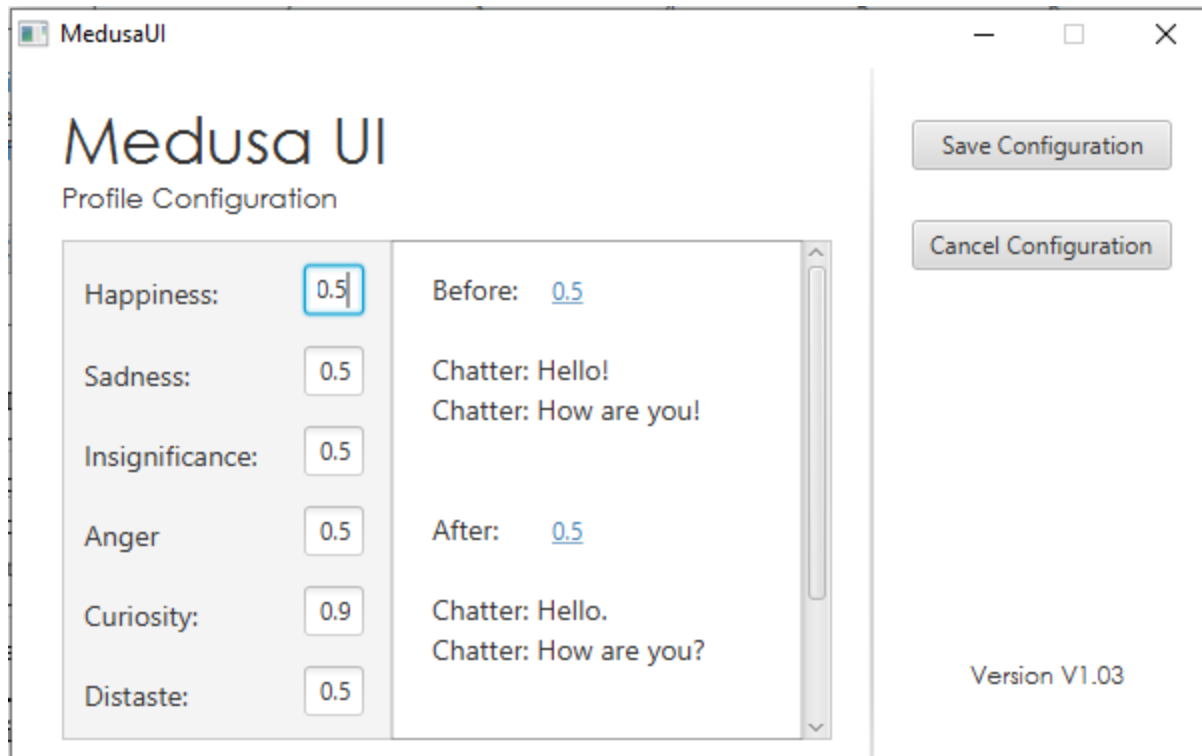
Profile: Athena  
Type: PRESET  
Configuration: PRESET configuration

Create Profile Configuration

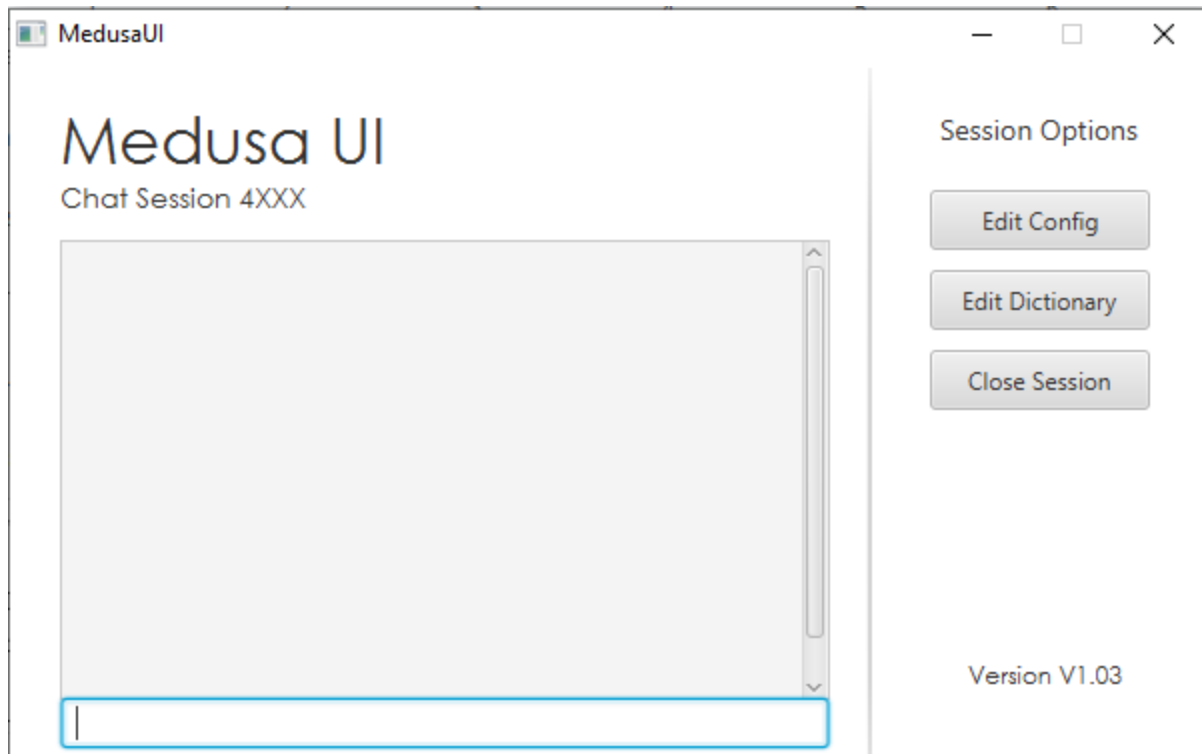
Not seeing a profile you like? Create a new profile here!

Version V1.03

The user should now be at the "Profile Configuration" screen. A custom query was designed to retrieve the configuration associated with profile, and is listed in the appendix as *Figure A5*. To see the various changes that a configuration multiplier has on the profile's expected behaviour, the values in each textbox can be changed. The user is expected to use increments of 0.1 for values ranging from 0.1 to 1.0. To see the current configuration, simply press the enter key on the keyboard, and the sample screen should present the respective value. As part of the sample input, press the enter key on the "Happiness" text field.



The "before" prompt is meant to show default values that the program takes on for "upper bounded" values. Having a value above 0.5 will apply the effect, however at 0.5 the program currently considers the effect to be weak and not apply the "positive" effect of the modifier. Here, the "After" field shows a less happy version of the default settings, and so the behaviour is accurate. In this version the logic does hold, as the comparison method uses the "Double.compare()" method, and values where the previous input and new input are the same are treated as applying a "negative" effect of the modifier. From here, press the "Save Configuration" button to continue.

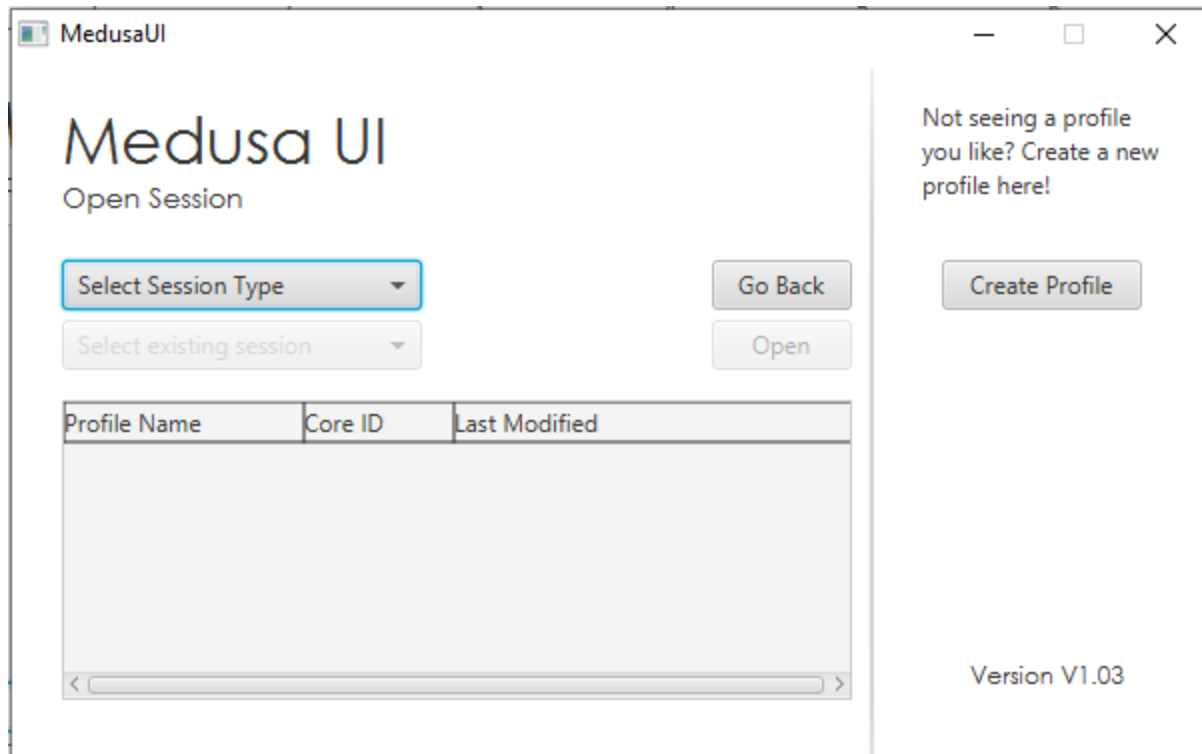


The user should now be at the "Chat Session" screen. This is the end of the create session complete flow. Click the "Close Session" button to return to the home screen.

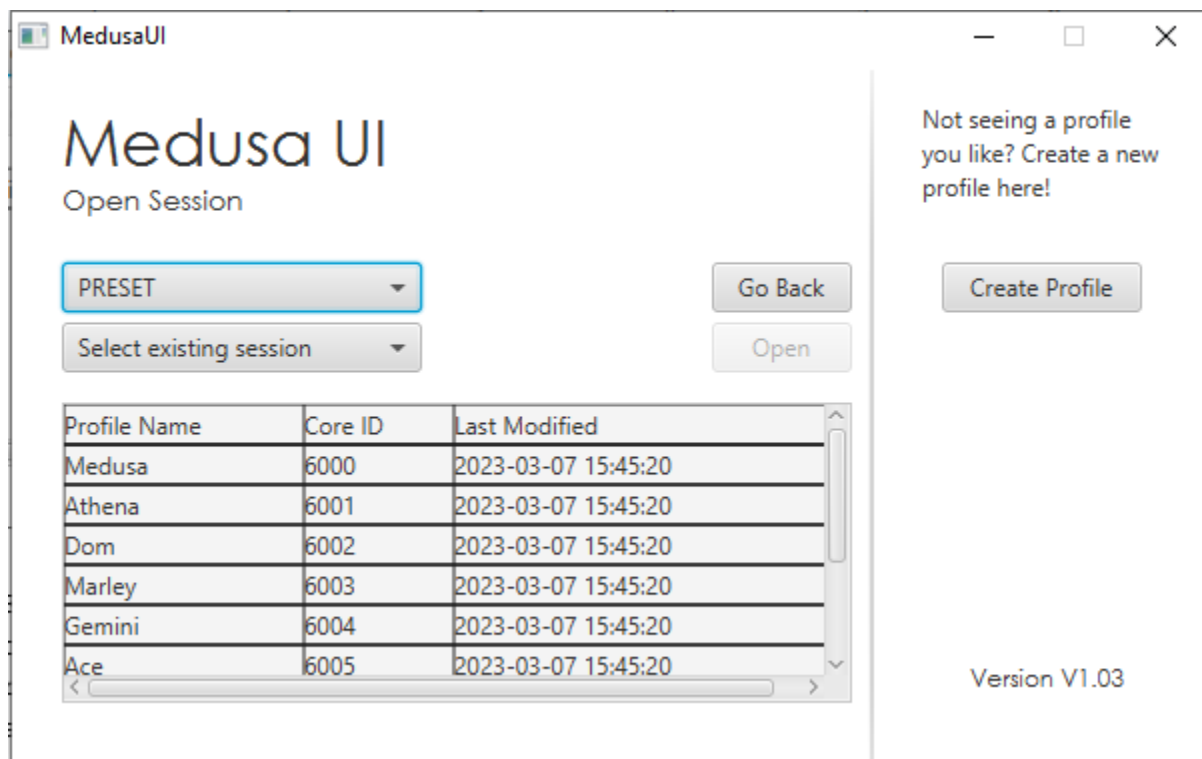
---

## 2.4

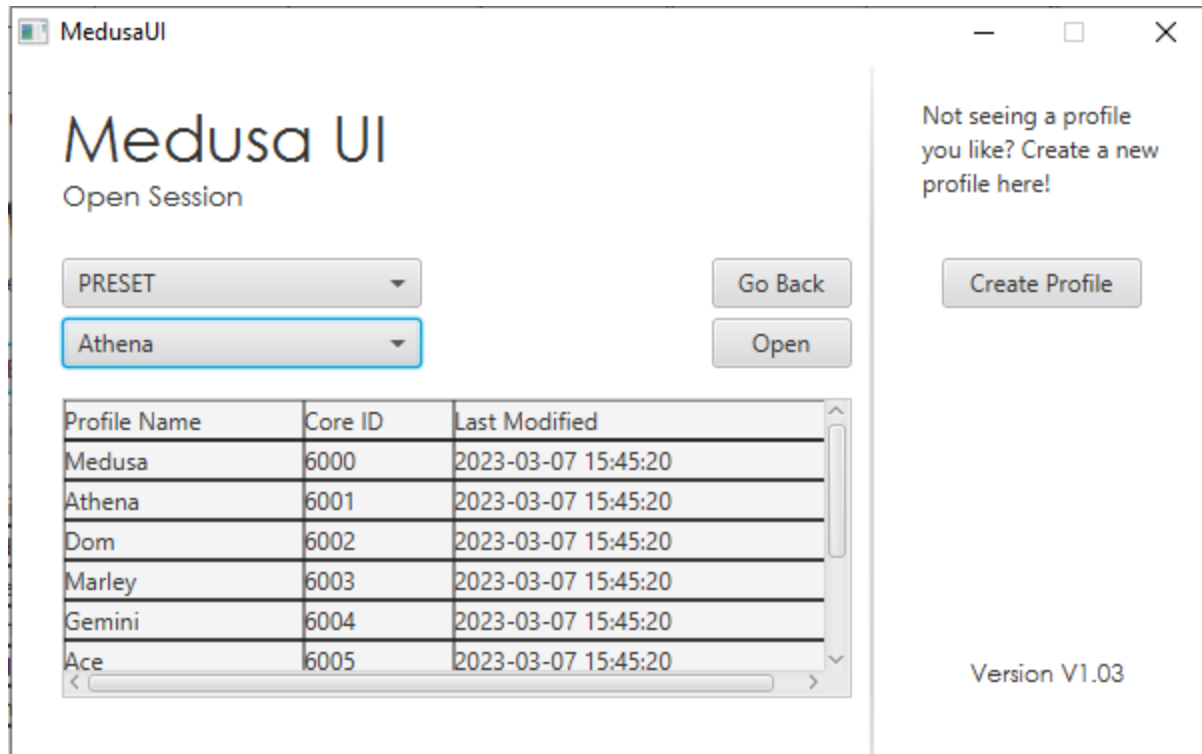
The open session flow begins with the user pressing the "Open Session" option.



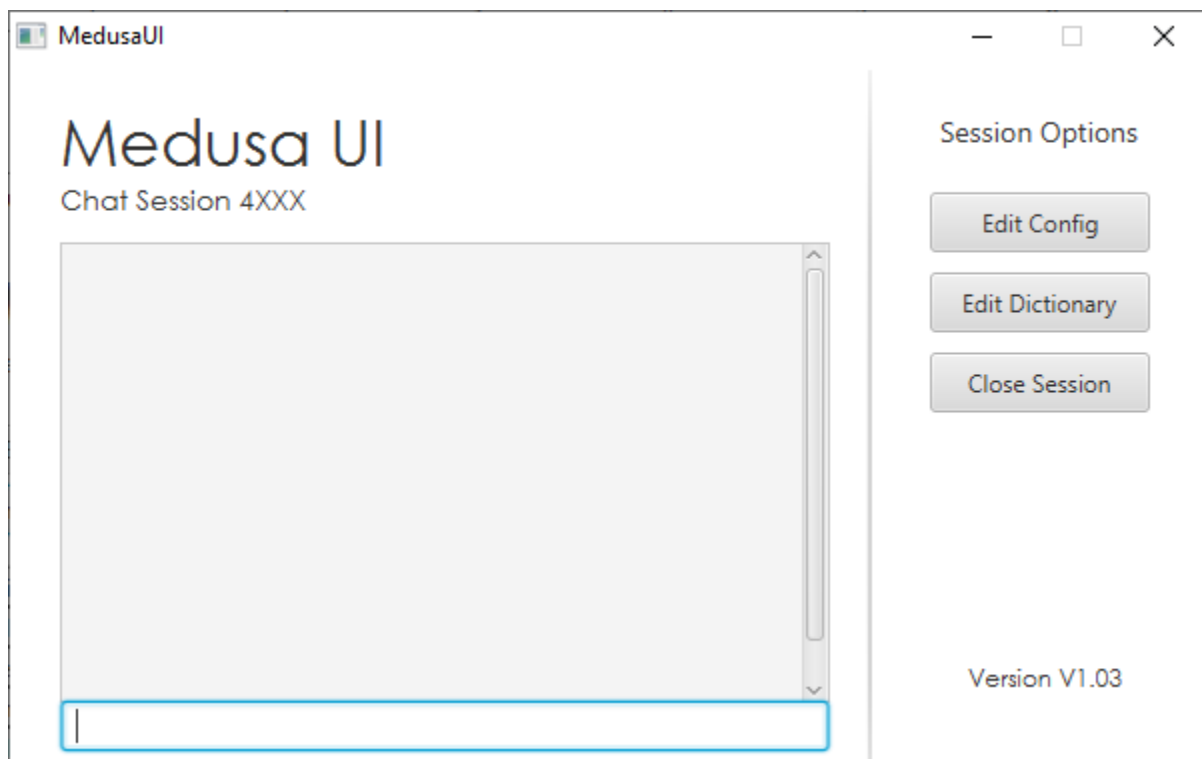
From here, press the "Select Session Type" option and select "PRESET" to load in a list of existing preset sessions. All preset profiles come with a session already created, so they will always exist.



The screen should now load in a list of the preset values as well as their respective "Core ID" and "Last Modified" datetimes. Core ID is used to identify a profile on a more specific level, as there could exist multiple profiles of the same name, however the Core ID allows a user to identify the original profile from a fork of an existing profile. To load in the profiles a special query is used that combines the "profile", "config", and "change\_log" entities as well as the joining "defines" and "updated\_to" relationships. This query is demonstrated in the appendix as *Figure A4*. As part of the sample input, select "Select existing session" and select "Athena".



From here, the "Open" button should now be available. Press the "Open" button to progress to the "Chat Session" screen.

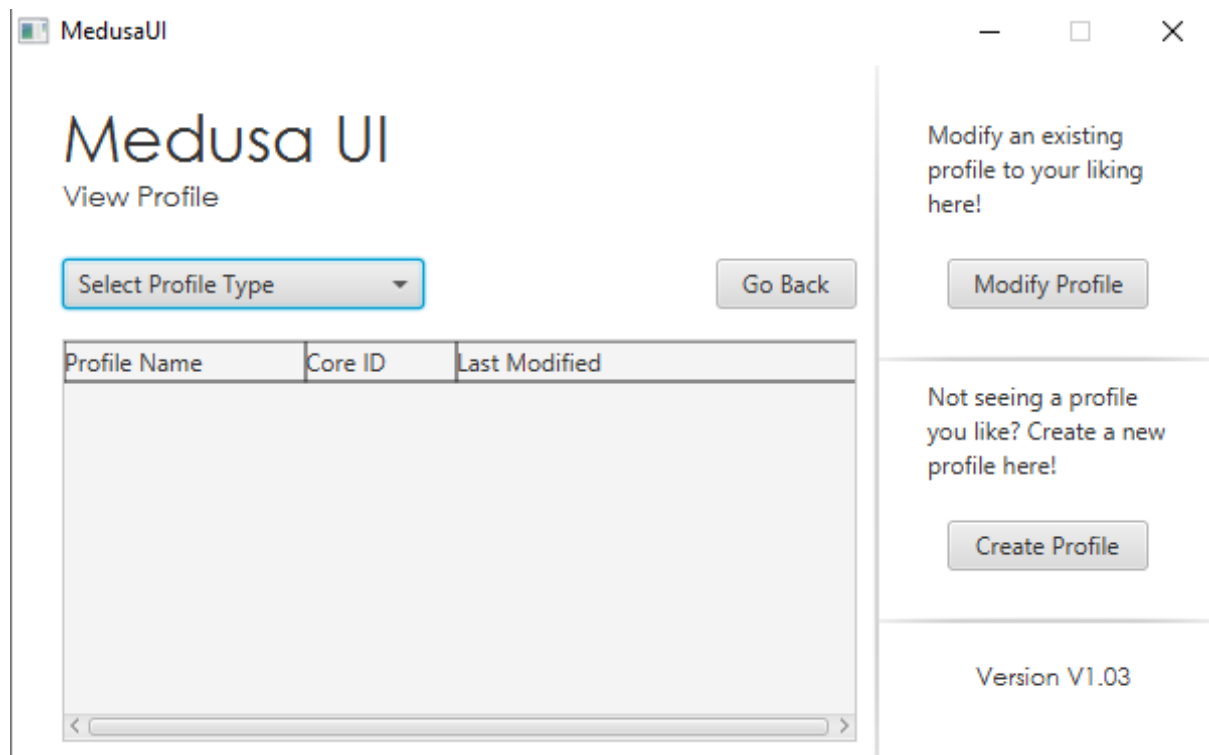


The user should now be at the "Chat Session" screen. This is the end of the open session flow. Click the "Close Session" button to return to the home screen.

---

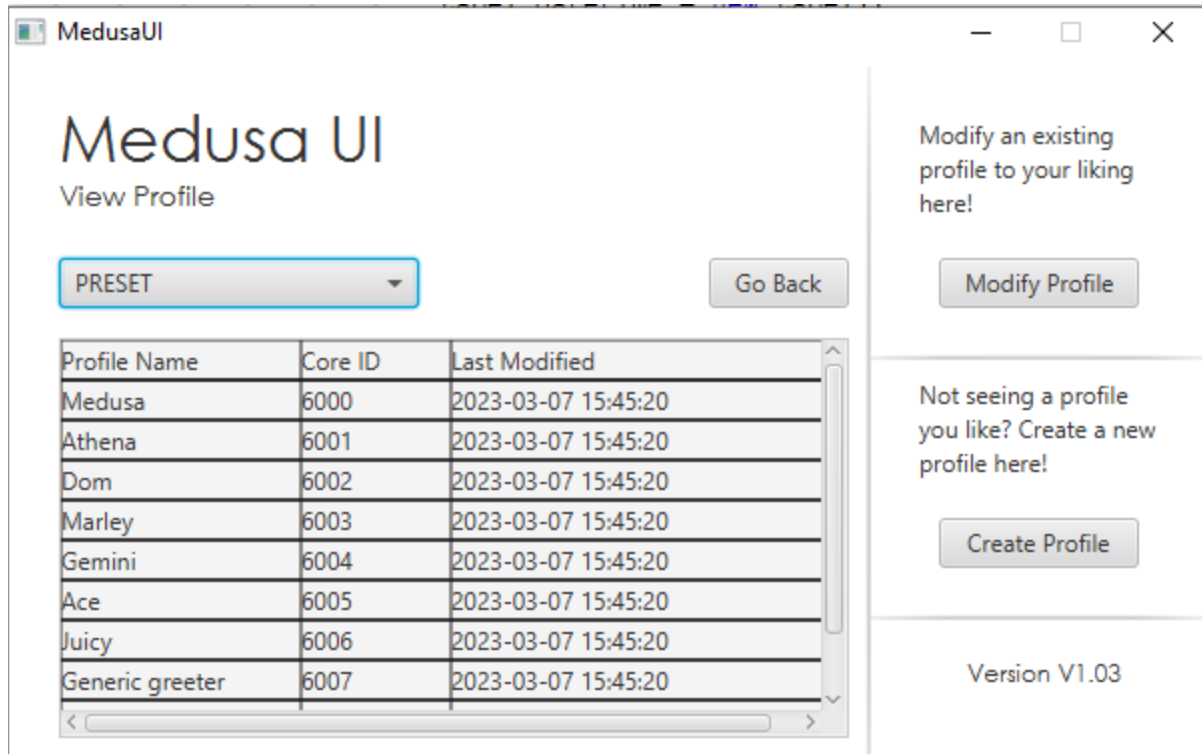
## 2.5

The view profile flow is extremely basic. The purpose of this flow is to show a list of either preset or custom users, and then allow the user to perform a set of operations to either modify a profile, create a new profile, or to simply return to the home screen. To start the view profile flow, press the "View Profiles" button.



Since only preset profiles exist, select the "PRESET" option from the "Select Profile Type" combobox.



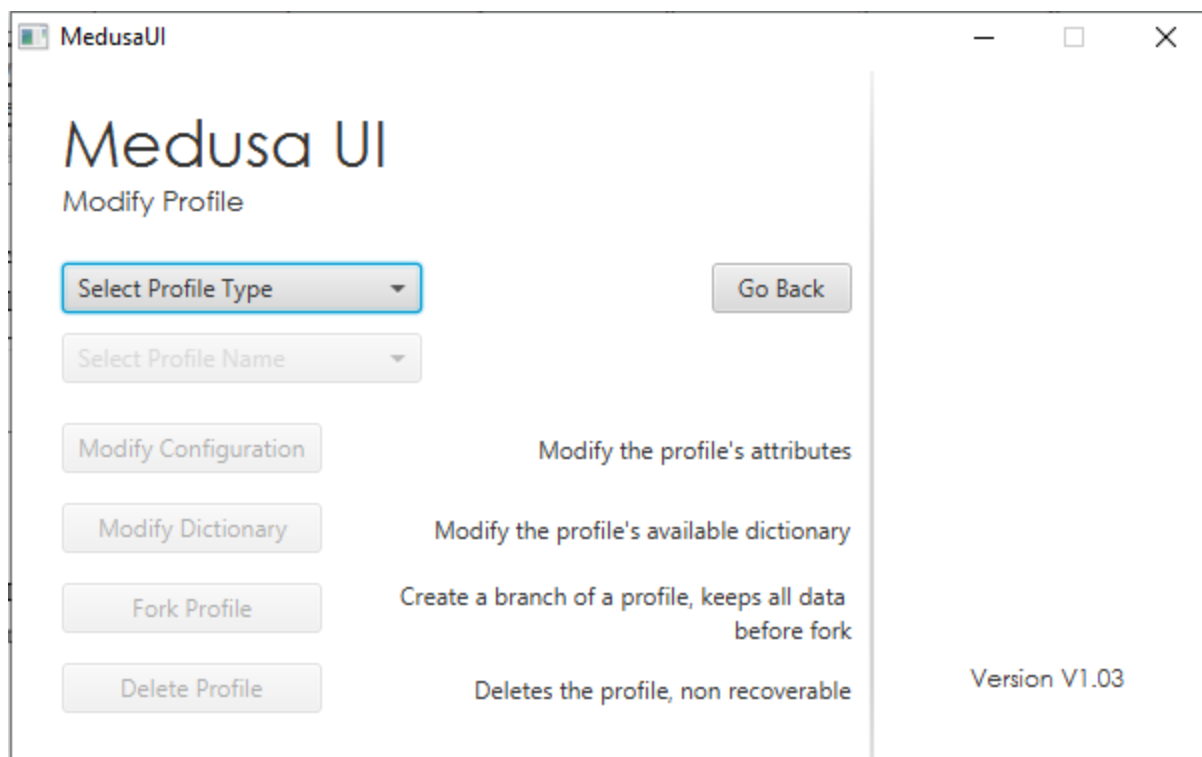


A list of all the presets should now be displayed. The view profile flow is now complete. Press the "Go Back" button to return to the home screen.

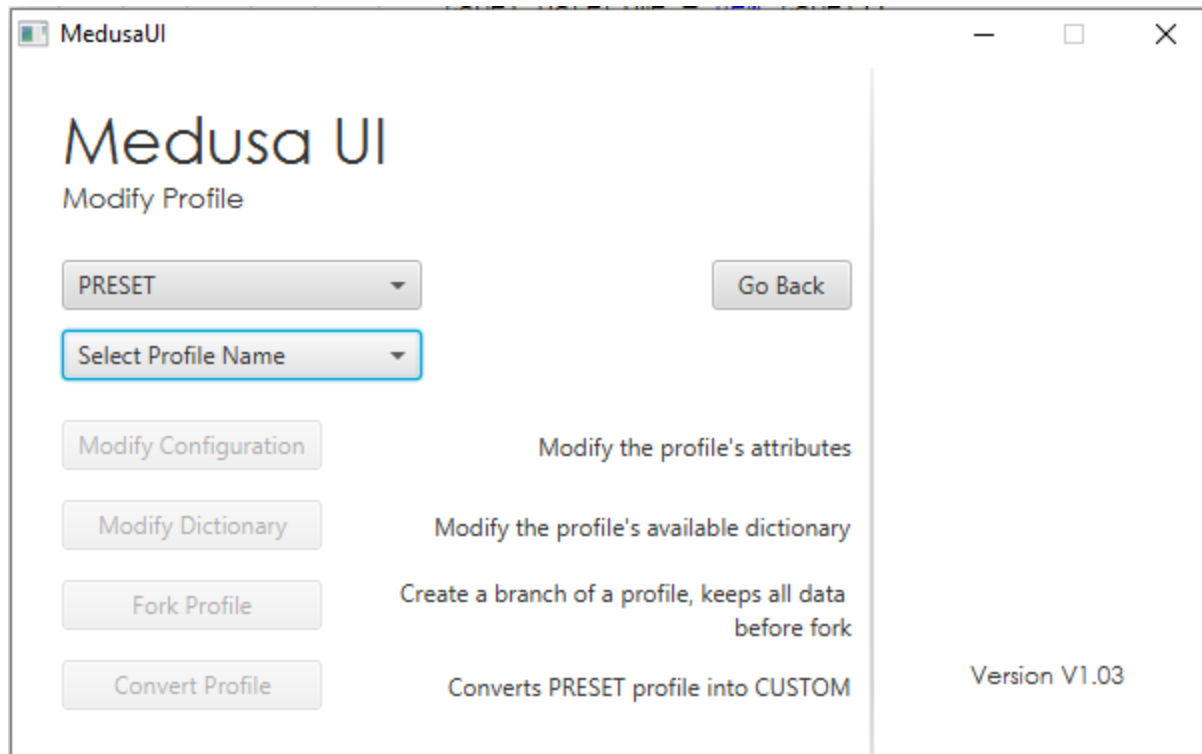
---

## 2.6

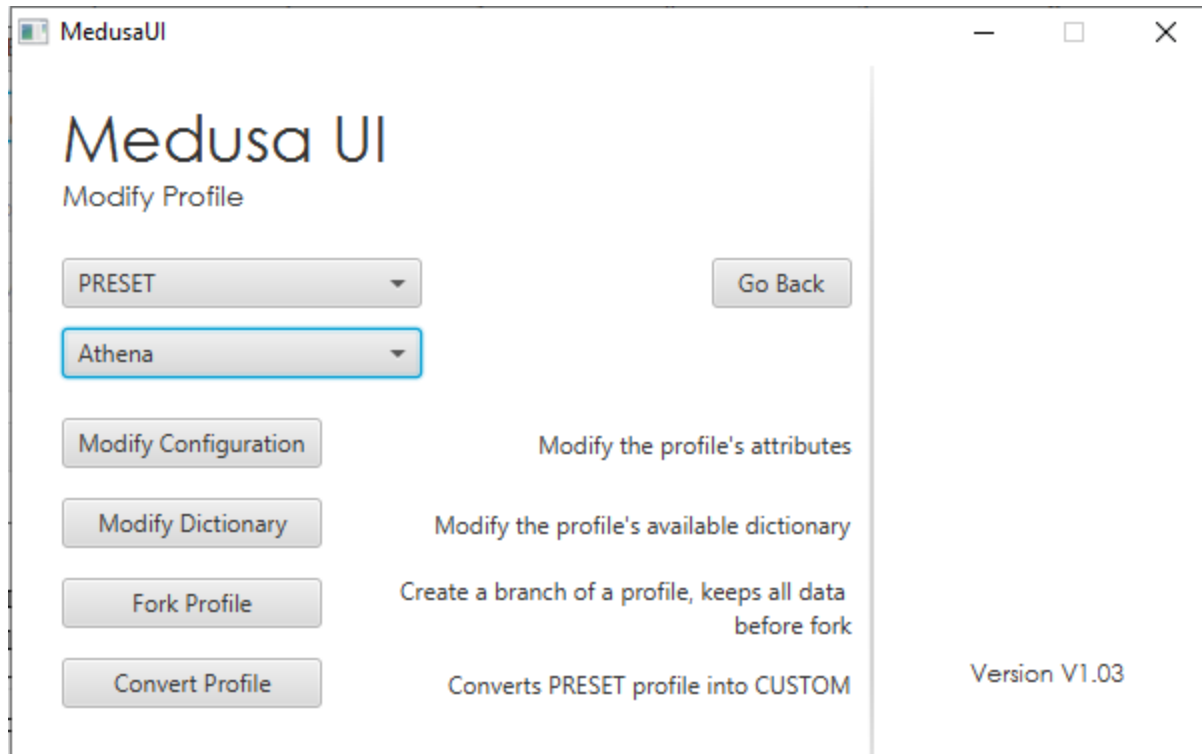
The modify profile flow is the most complex flow currently implemented in the current build. Repeat the previous steps until the user reaches the "View Profile" screen. From there, press the "Modify Profile" button.



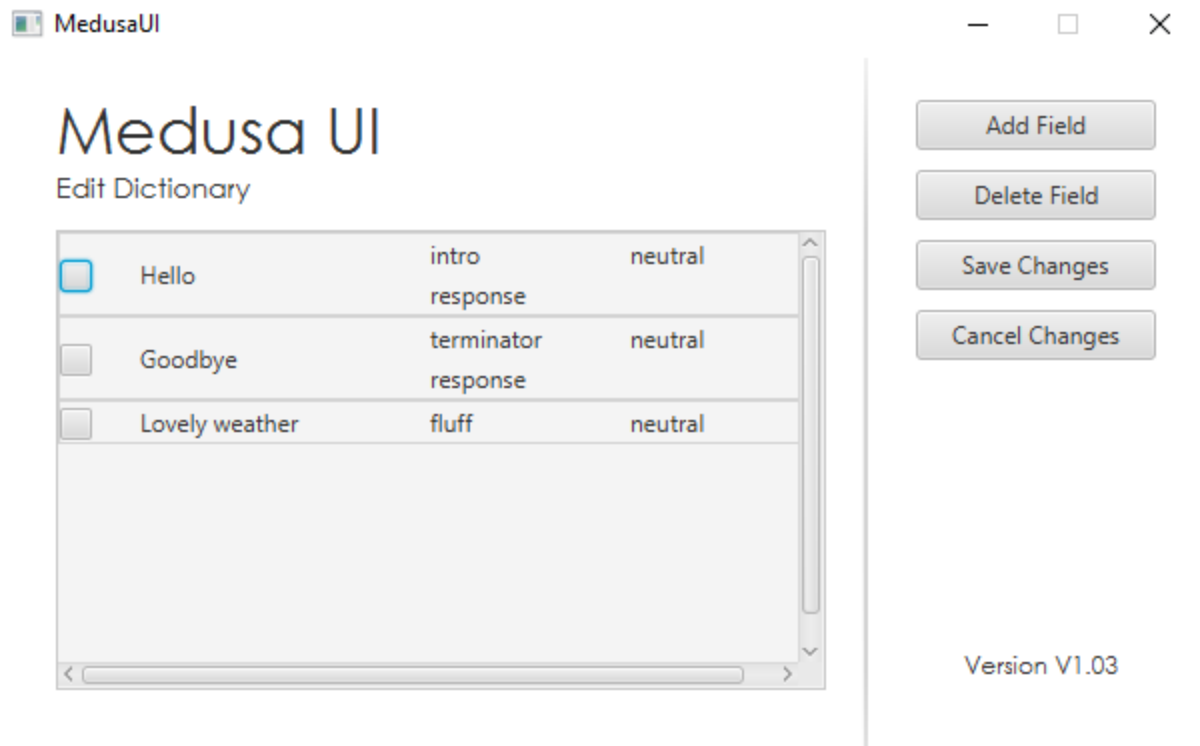
From here a user has a lot of options to modify a profile. The preset and custom profile types are different in that custom profiles can be deleted, however preset profiles can only be converted to a custom profile. All profile types can be forked, and the generic modifiers to modify the configuration and modify the dictionary exist for all profile types as well. Select the "Select Profile Type" combobox and select the "PRESET" option.



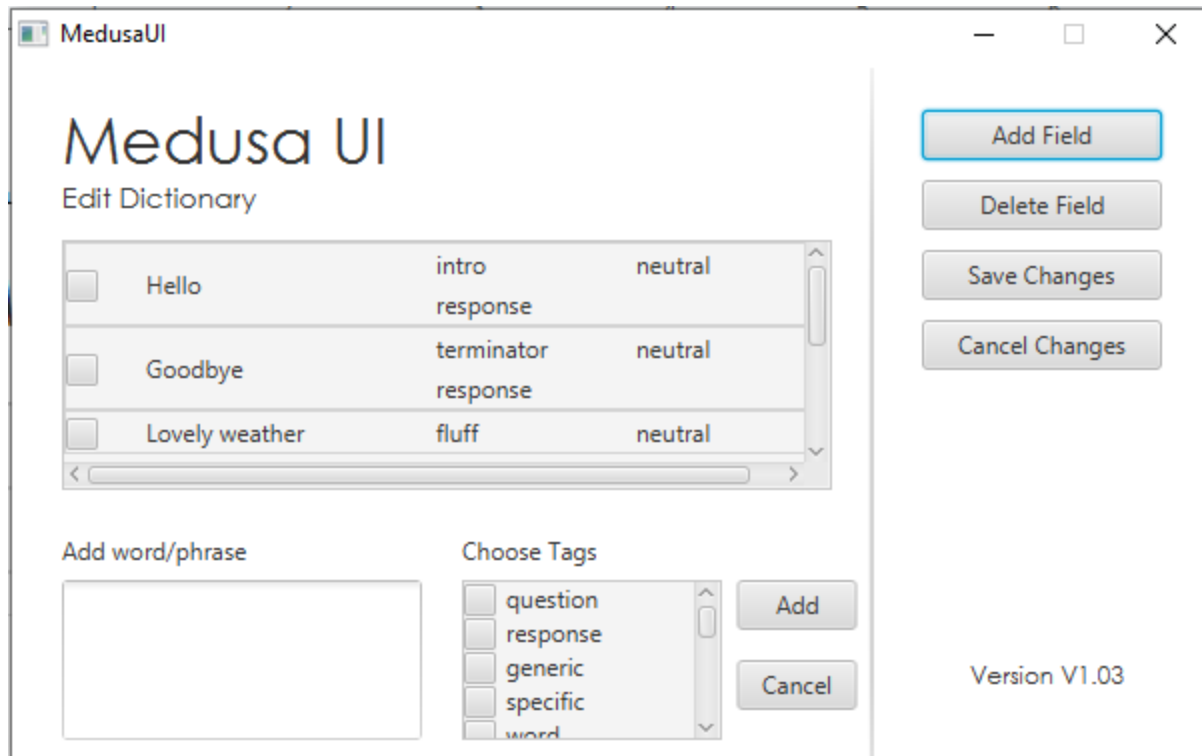
The "Delete Profile" option that appeared as part of the default screen should now be replaced with the "Convert Profile" option. As part of the sample input, select "Athena" from the "Select Profile Name" combobox.



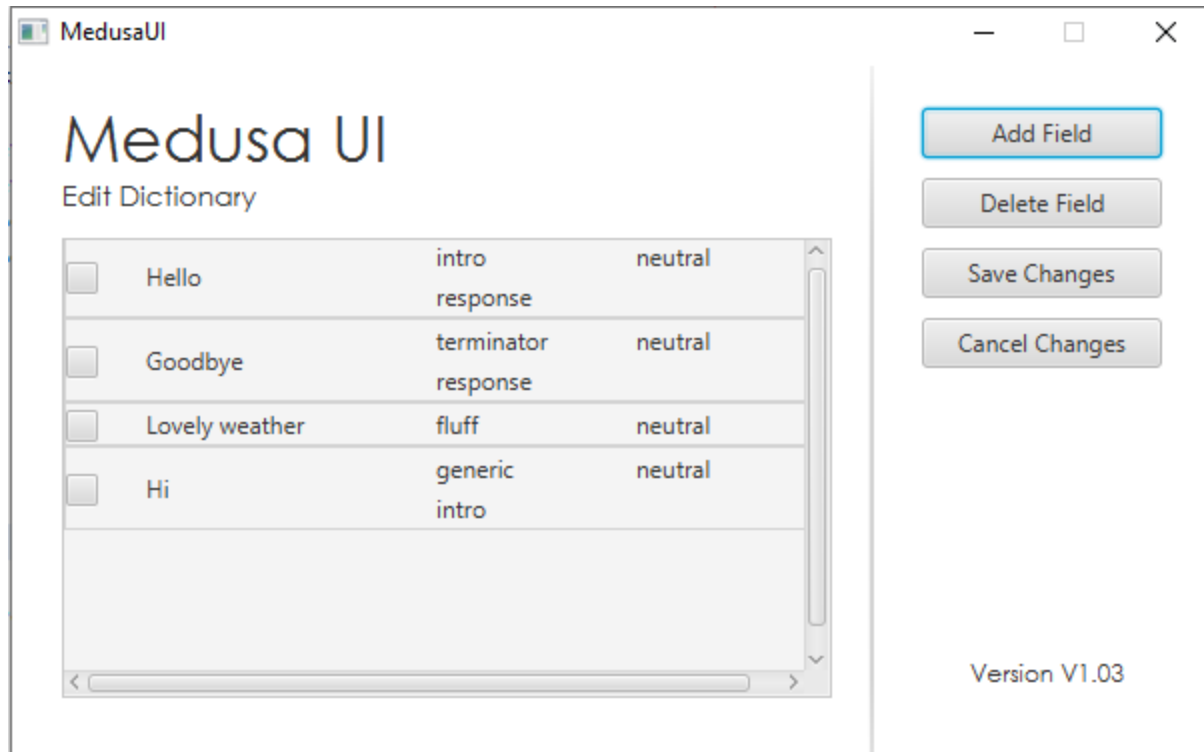
From here all of the options should become available. All of the options work at a local level and a small sample will be provided for each starting from "Modify Dictionary". Click the "Modify Dictionary" button to begin the edit dictionary flow.



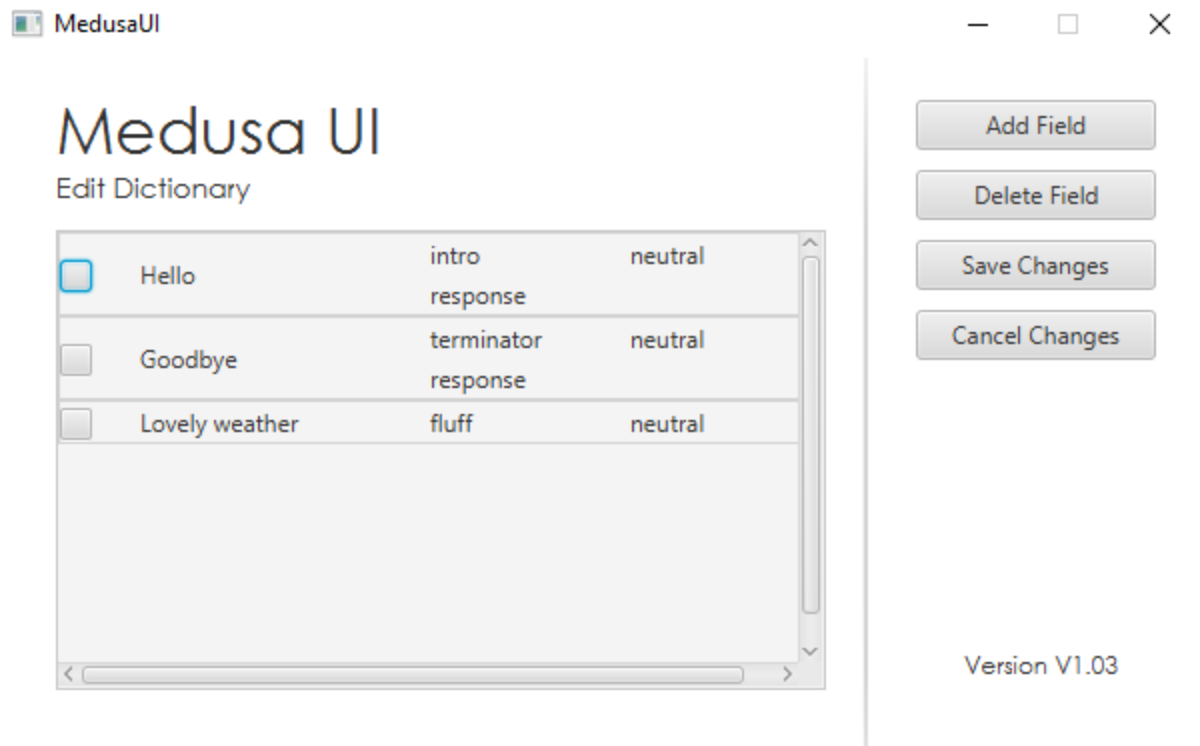
The edit dictionary screen displays the dictionary of a profile and opens it to the option of modification. A custom query has been made to read in the associated dictionary of a profile, and it is demonstrated as *Figure A6* in the appendix. Selecting a checkbox next to an entry field allows the user to delete it, however the changes are not made final until the user presses the "Save Changes" button. Pressing "Cancel Changes" returns the user to the previous screen, undoing all local operations. To demonstrate the local add field functionality, press the "Add Field" button.



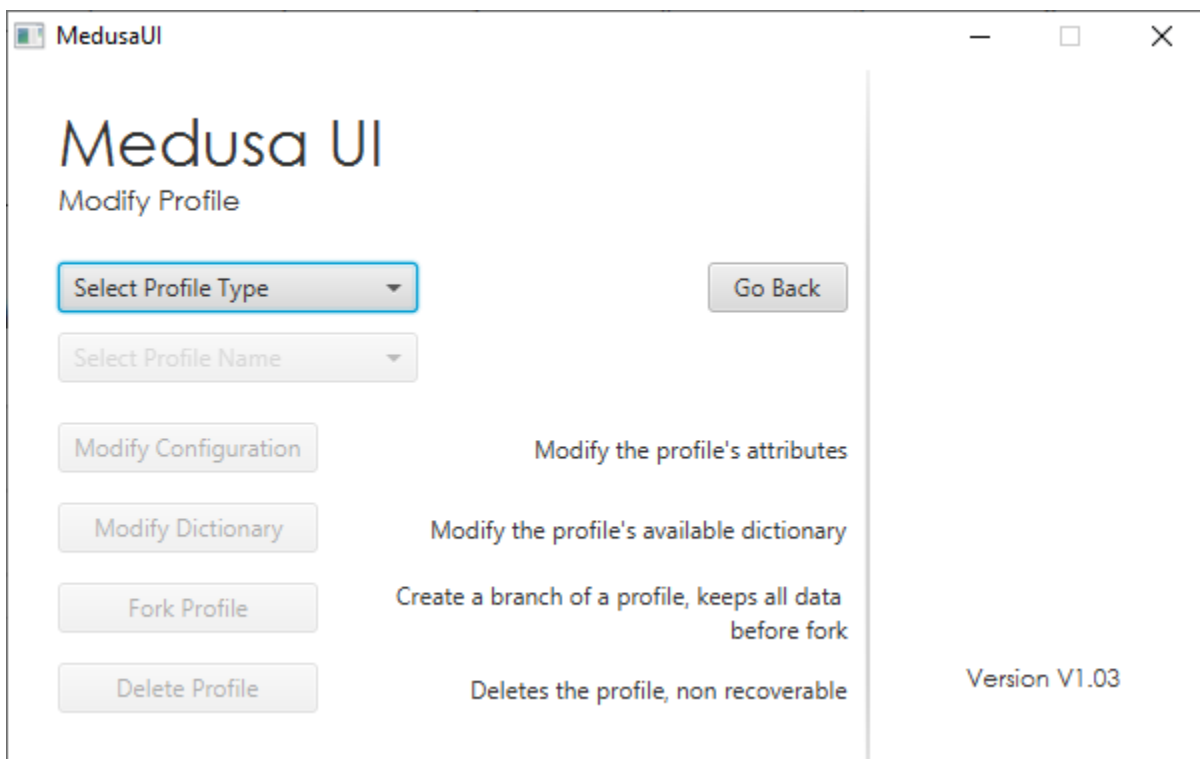
The screen now displays the interface to add a new entry. Entries entered may be either a word or a phrase, and entirely up to the user. The user must also select the context tags and emotional inflection tags associated with the entry to allow the profile to understand the meaning behind the entry. Context tags are all checkboxes ranging from "question" and ending at "emotional". Emotional tags range from "negative" and extend until the end of the list. It is recommended to add at least 1 of each type. As part of the sample input, enter "Hi" as the entry, select "generic", "intro", and "neutral" as the tags, and then press the "Add" button.



The new "Hi" entry should now appear as part of the list, showing that it has been added on a local level. Next, select the checkbox beside the "Hi" entry, and press the "Delete Field" button.



The "Hi" entry has now been locally removed. To finalise the changes, press the "Save Changes" button.





The user should now be back at an empty "Modify Profile" screen. Re-enter "PRESET" and "Athena" in the respective comboboxes, and now press the "Fork Profile" button.

Medusa UI

Fork Profile

Select Core ID

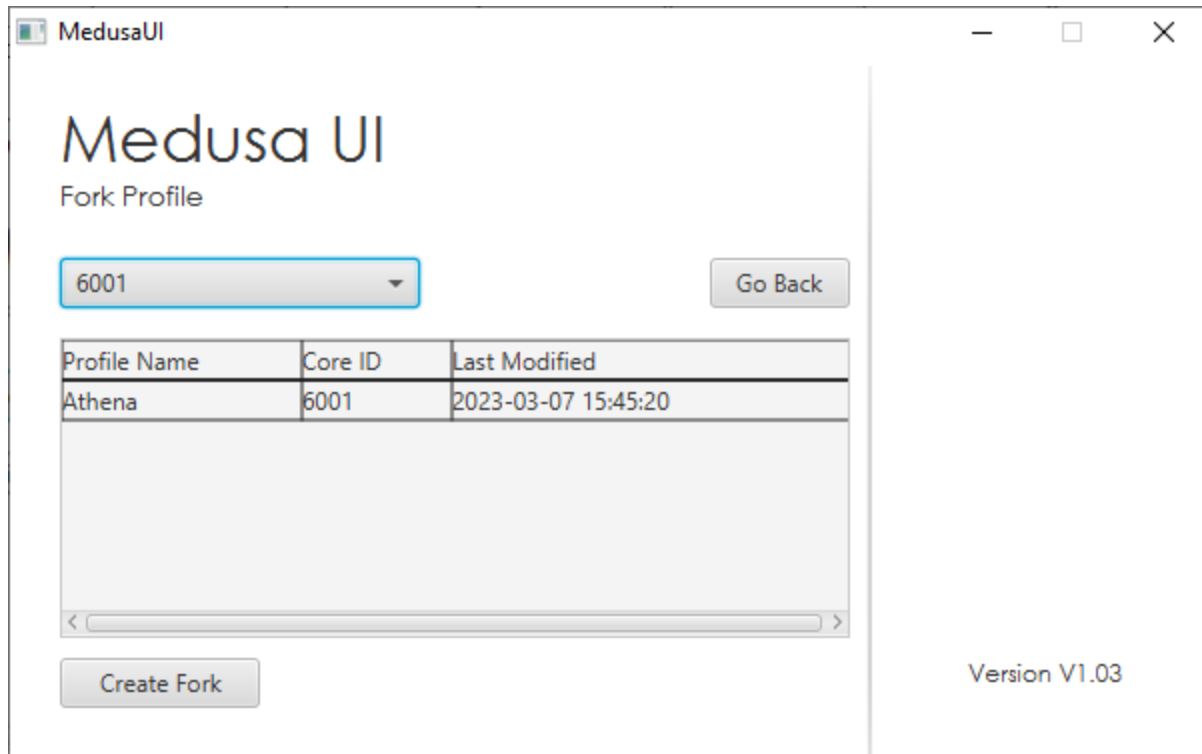
Go Back

Profile Name	Core ID	Last Modified
--------------	---------	---------------

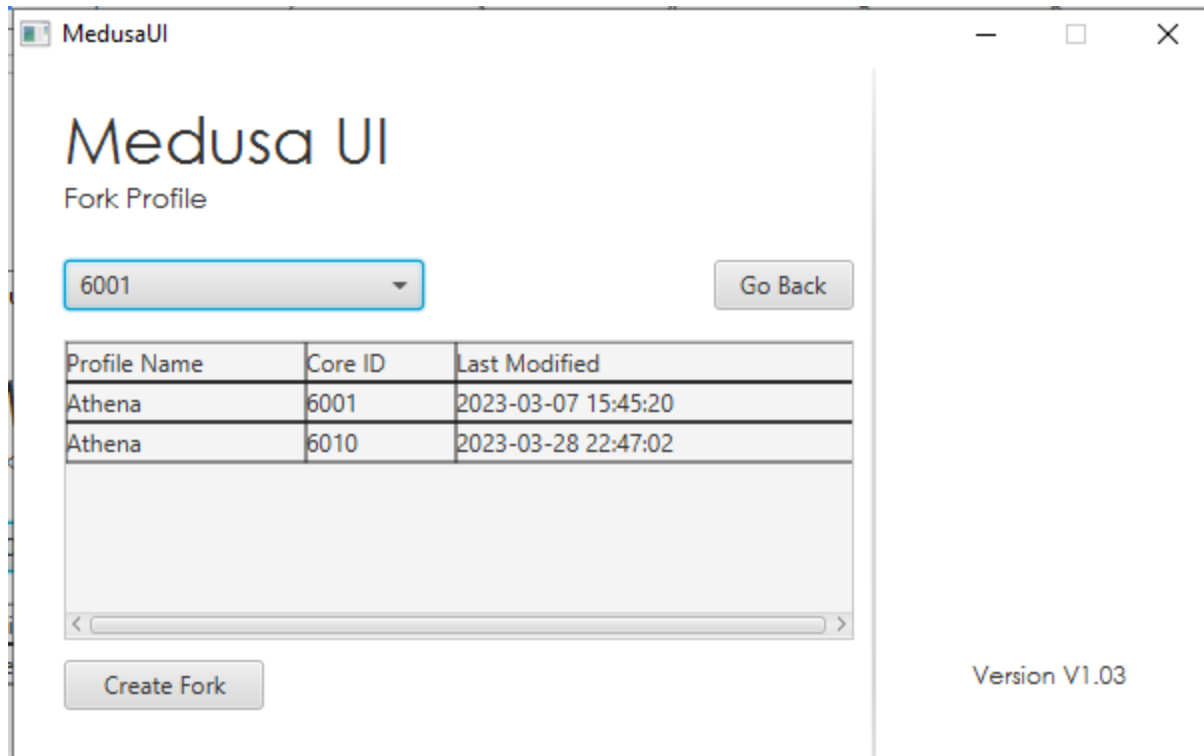
Create Fork

Version V1.03

From here the "Fork Profile" screen becomes available. In the "Select Core ID" combobox, a list of all related CoreIDs that relate to a profile of the "Modify Profile" screen's selection are loaded and displayed. There should only be 1 option available, however, so select it.

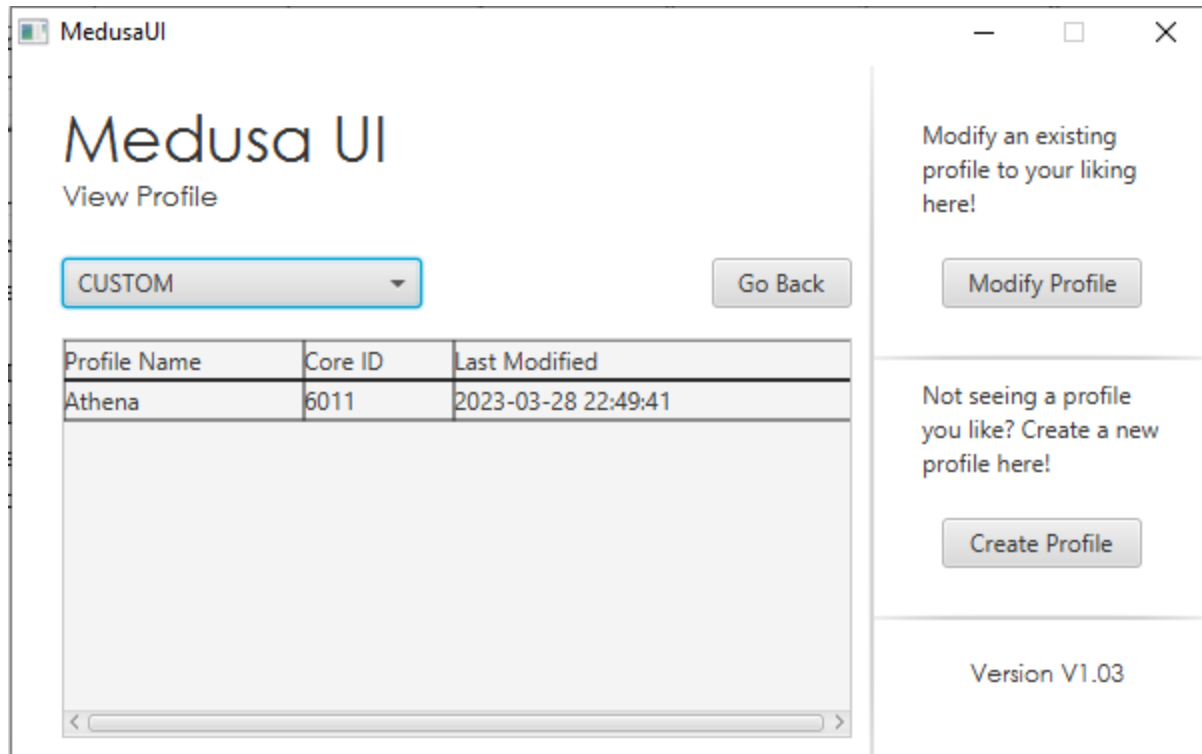


Since "6001" is the only core of the profile, press the "Create Fork" button to create a fork of the "Athena" profile. Forks are meant to be used as a means to create a new session for a profile while keeping all previous made changes to the profile including the dictionary. This allows users to create multiple conversations with a profile while modifying smaller details to see the different paths a profile may take while interacting with a user. Once the "Fork Profile" screen reloads, select "6001" from the dropdown.



A new core with ID "6010" should appear. This means that a new profile fork was created with a new core ID corresponding to this newly created session. It can now be loaded and viewed in the "View Profiles" screen. This is the end of the fork profile flow, to return to the modify profile flow press the "Go Back" button. Once at the "Modify Profile" screen, Re-enter "PRESET" and "Athena" in the respective comboboxes, and now press the "Convert Profile" button.

The screen should have reset once more, meaning that the profile has been successfully converted to a "CUSTOM" type. This change can now be viewed in the "View Profiles" screen, with the "Select Profile Type" being selected to "CUSTOM".

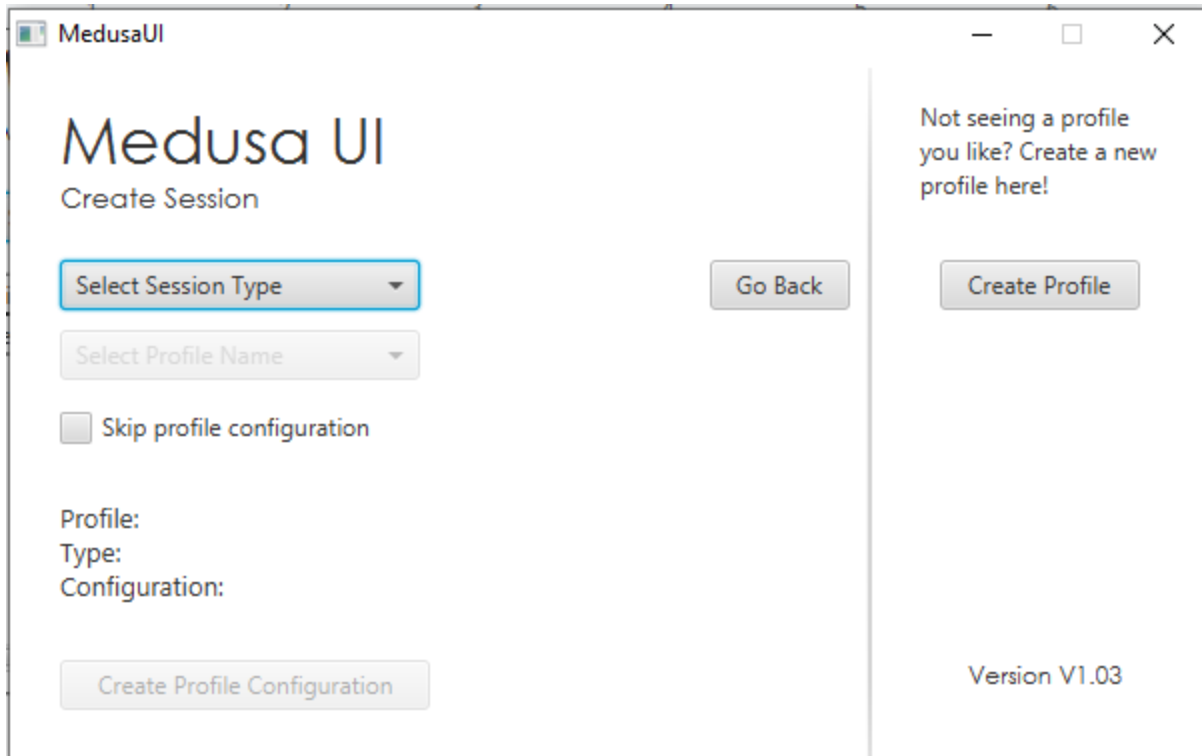


This is the end of the modify profile flow. Press the "Go Back" button to return to the home screen.

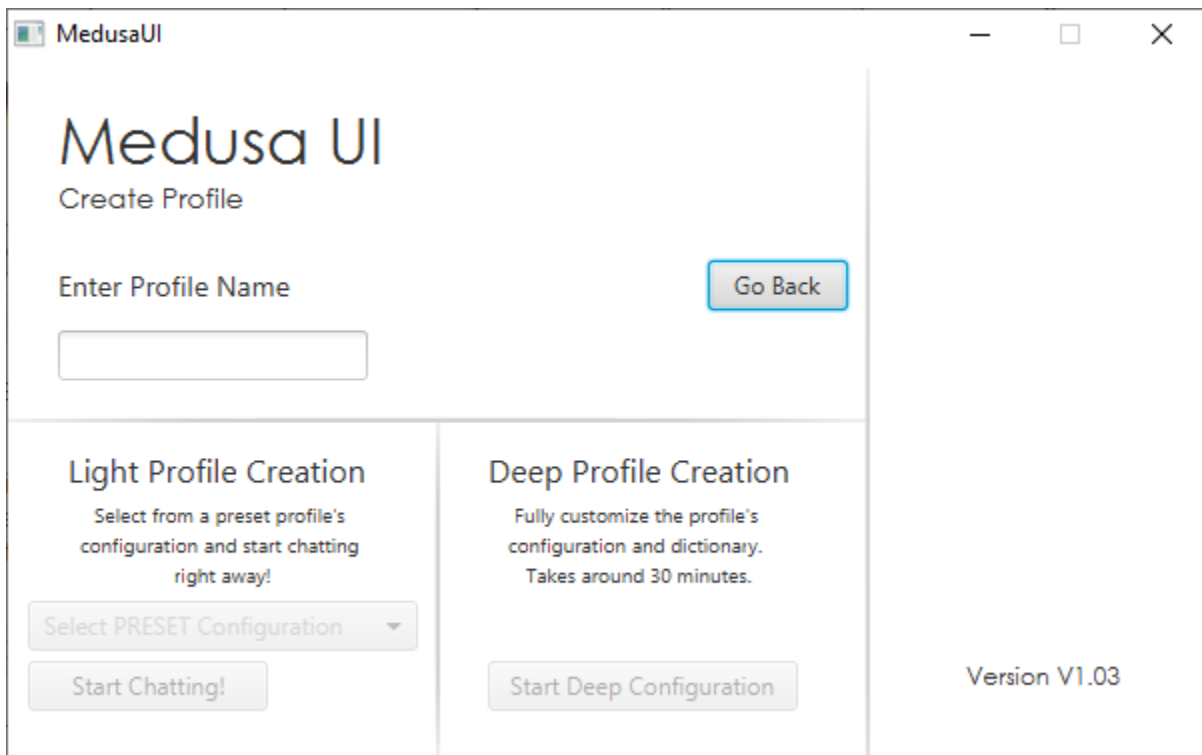
---

## 2.7

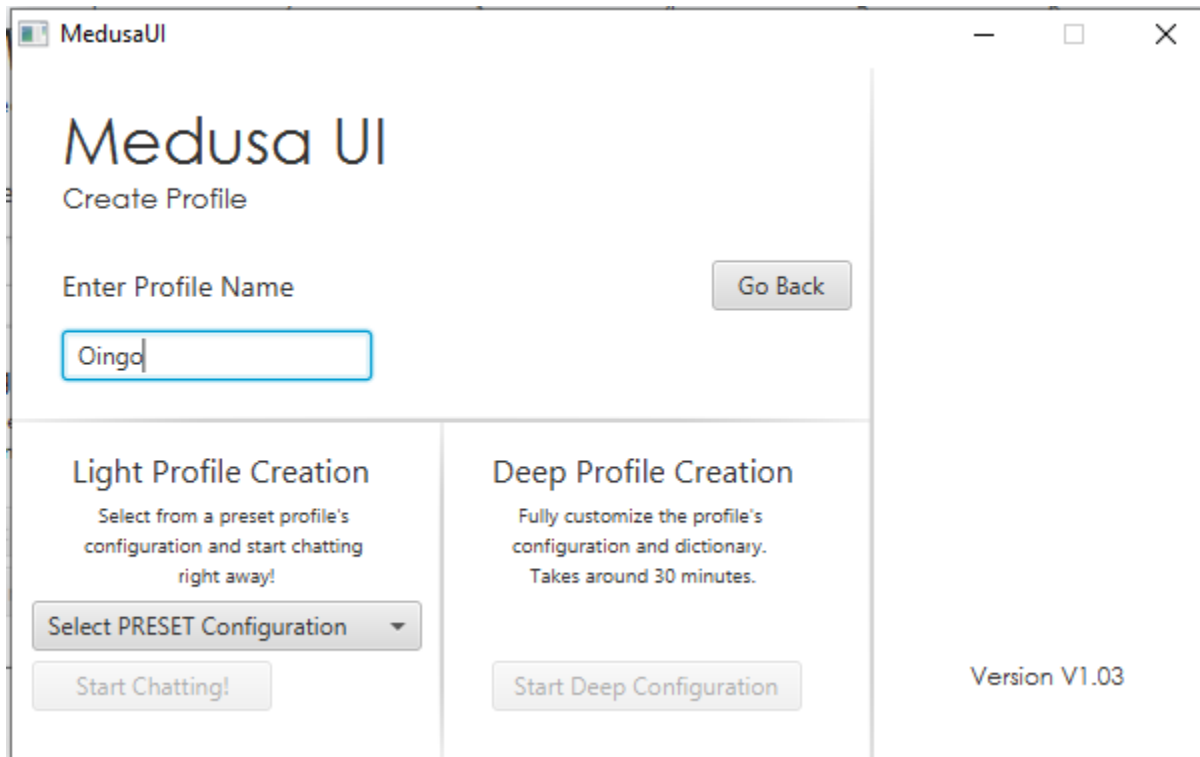
The last option of this build is the "Light Creation" flow. This flow allows for the user to quickly start chatting with a profile by creating a new custom profile and loading the configuration of an existing preset profile type. To enter the light configuration flow, press the "Create Session" or "View Profile" buttons. As part of the sample input, the "Create Session" button will be selected.



From the "Create Session" screen, press the "Create Profile" button to progress.



The current version of the program displays prompts for a light and deep profile creation option, however only the "Light Creation" option is partially implemented. To progress, enter a profile name in the "Enter Profile Name" prompt. Do note, that if a profile that already exists is entered, the prompt will block any further input until the error is corrected. As part of the sample input, enter "Oingo" in the "Enter Profile Name" text field, and press enter.



MedusaUI

# Medusa UI

Create Profile

Enter Profile Name

Oingo

Go Back

### Light Profile Creation

Select from a preset profile's configuration and start chatting right away!

Select PRESET Configuration

Start Chatting!

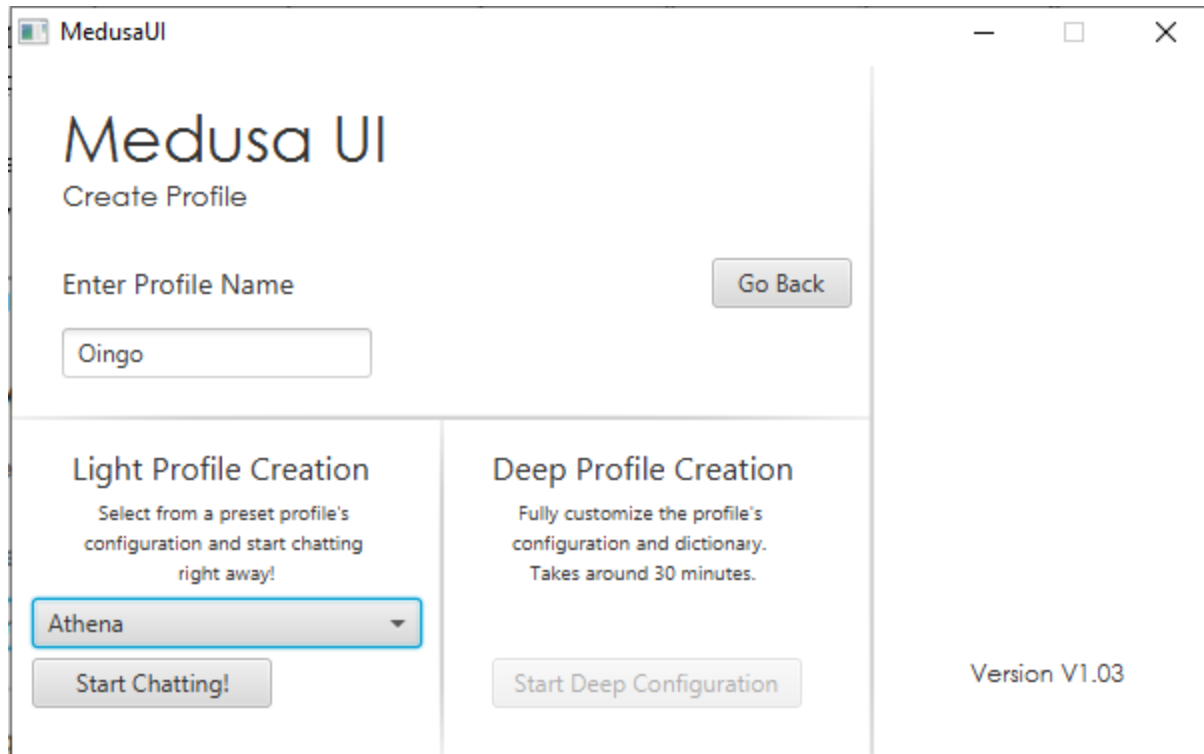
### Deep Profile Creation

Fully customize the profile's configuration and dictionary. Takes around 30 minutes.

Start Deep Configuration

Version V1.03

From here, the preset configuration option will be enabled. This list contains all the existing profiles who's configuration can be linked to the new profile. As part of the sample input, select "Athena".



The image shows a window titled "MedusaUI" with standard window controls (minimize, maximize, close) in the top right corner. The main heading is "Medusa UI" followed by the subtitle "Create Profile". Below this is a label "Enter Profile Name" and a text input field containing the text "Oingo". To the right of the input field is a "Go Back" button. The interface is divided into two columns. The left column is titled "Light Profile Creation" and contains the text "Select from a preset profile's configuration and start chatting right away!". Below this text is a dropdown menu with "Athena" selected. At the bottom of this column is a "Start Chatting!" button. The right column is titled "Deep Profile Creation" and contains the text "Fully customize the profile's configuration and dictionary. Takes around 30 minutes." Below this text is a "Start Deep Configuration" button. In the bottom right corner of the window, the text "Version V1.03" is displayed.

MedusaUI

# Medusa UI

Create Profile

Enter Profile Name

Oingo

Go Back

## Light Profile Creation

Select from a preset profile's configuration and start chatting right away!

Athena

Start Chatting!

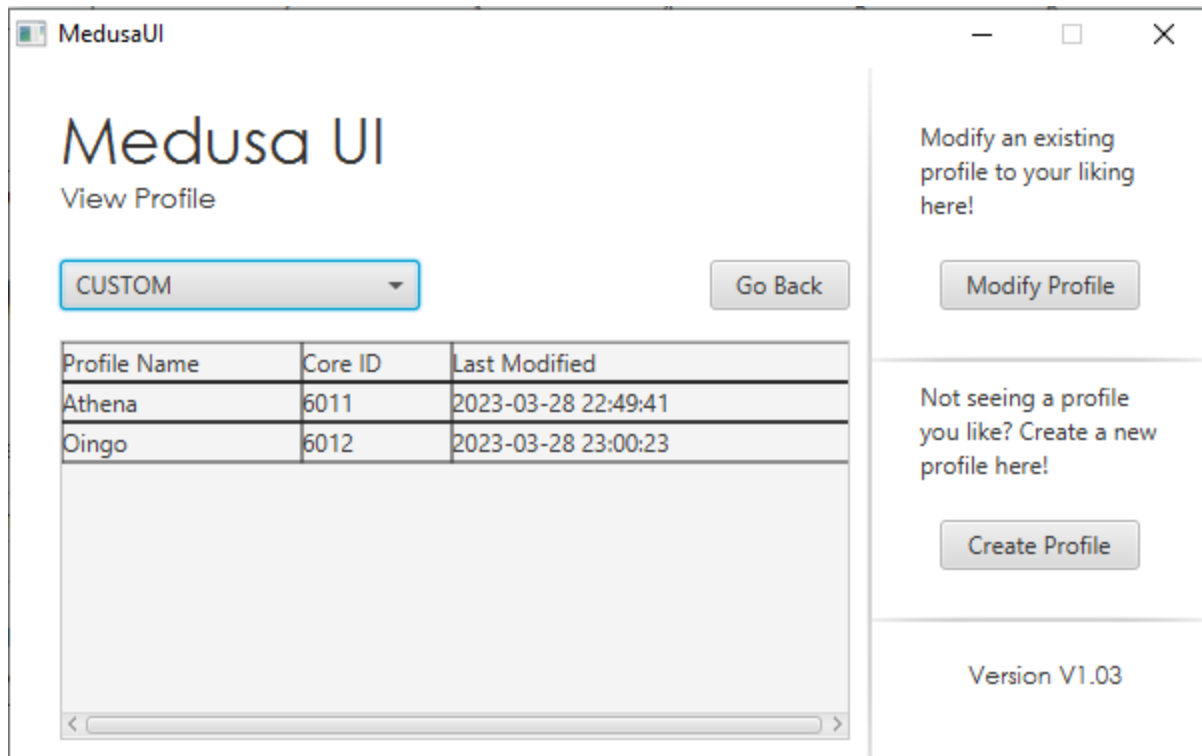
## Deep Profile Creation

Fully customize the profile's configuration and dictionary. Takes around 30 minutes.

Start Deep Configuration

Version V1.03

Finally, the "Start Chatting!" button should be enabled. Press the button to open up a chat session. Once the chat session is closed, the new "Oingo" profile will be available to be viewed from the "View Profile" screen.



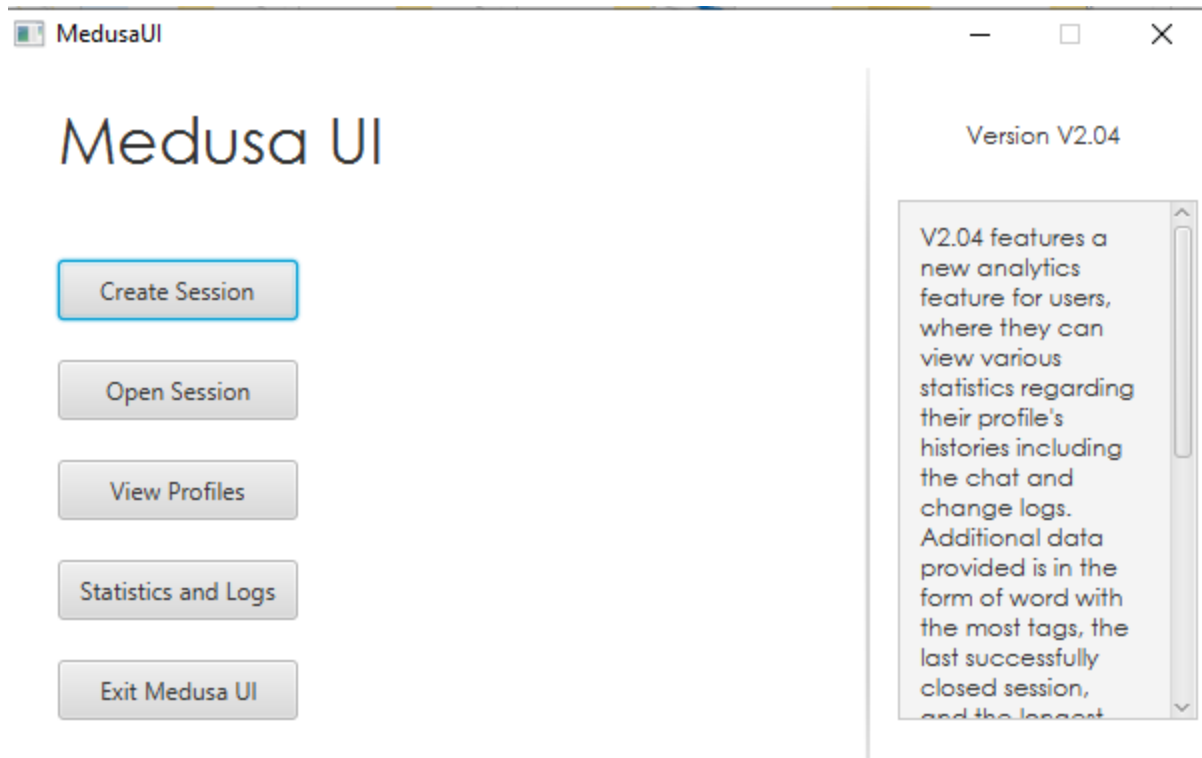
The light profile creation flow is now complete. There are no more flows to demonstrate in version 2.04 of the MedusaUI program.

---

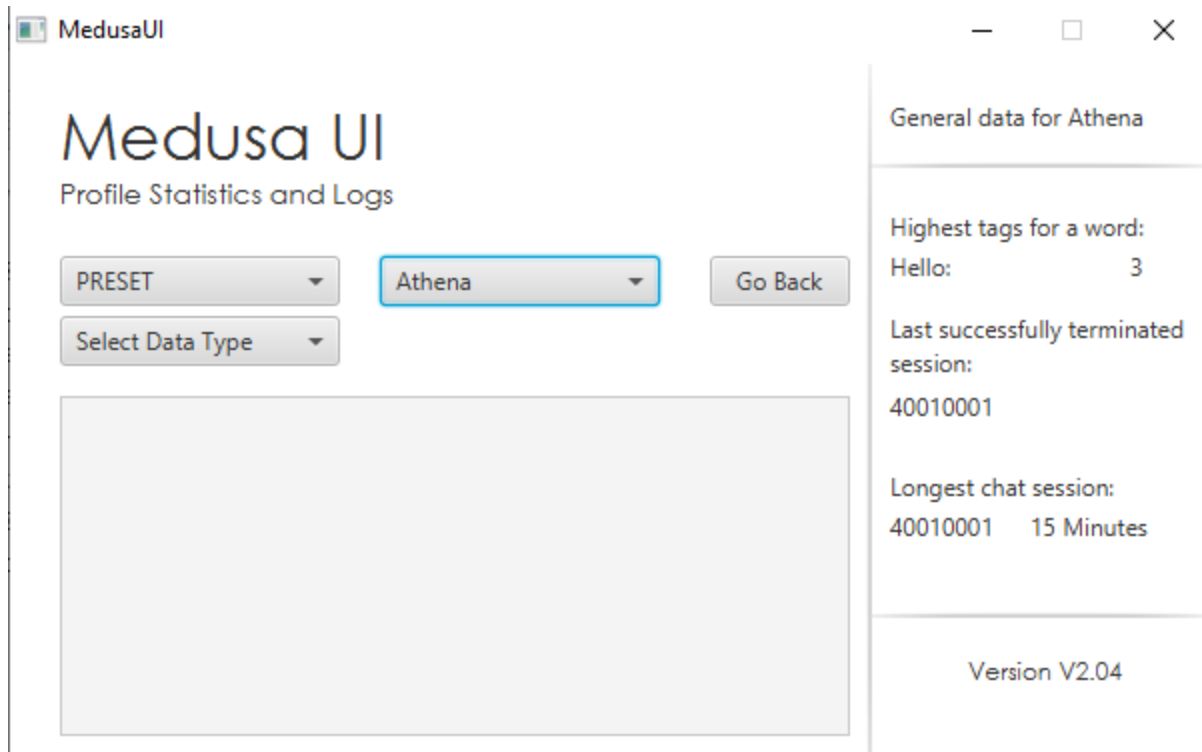
## 2.8

The newest feature of version 2.04 is the "Statistics and Logs" screen, easily accessible from the home screen via the "Statistics and Logs" button.





From here, the user is introduced to the commonly seen comboboxes for "Select Profile Type" and "Select Profile", however here the screen-specific combobox for "Select Data Type" is also present. The statistics screen is also incredibly query-heavy, with all of the data being present being drawn from one query or another. As part of the sample input, select "PRESET" for "Select Profile Type" and select "Athena" for "Select Profile".



Immediately some data should be presented on the rightmost panel. Here special queries found in the appendix as figures A7 - A11 have been utilised to either directly present data or assist with calculating data presented on the screen. Following this basic data presentation, the user may select whether they wish to review the change log or chat log for their respective profile. As part of the sample input, first select the "Change log" option from the "Select Data Type" combobox.

MedusaUI

Medusa UI

Profile Statistics and Logs

PRESET

Athena

Go Back

Change log

Change Log ID	Recorded Change	Change Type	Last Updated
70010001	Created personality	Profile creation	2023-03-07 1
70010002	Session Opened	Session update	2023-04-02 1
70010003	Session Closed	Session update	2023-04-02 2

General data for Athena

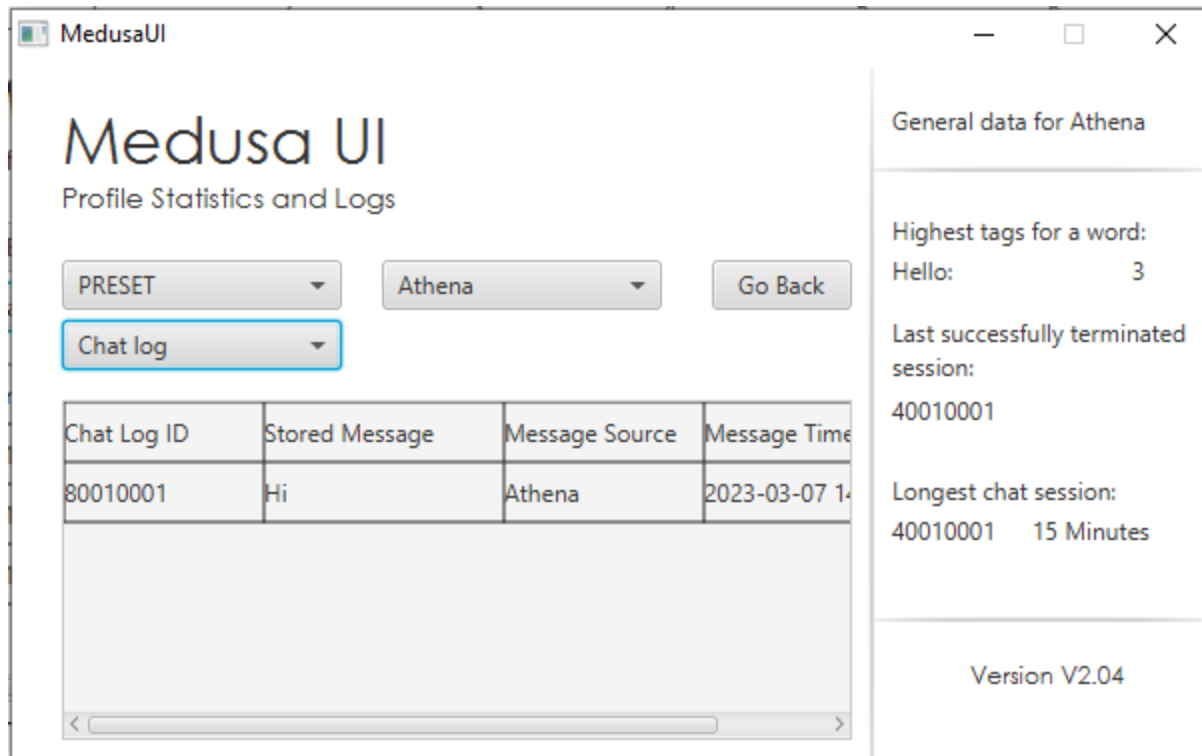
Highest tags for a word:  
Hello: 3

Last successfully terminated session:  
40010001

Longest chat session:  
40010001 15 Minutes

Version V2.04

Now demonstrated is the change log associated with the profile. All information from the database is pulled and filled in the same order as presented, as well as the time-date record associated with each change. Special query found under appendix *figure A12* has been used to generate this data. Now, select the "Chat log" option from the previously used combobox.



Now, the chat log associated with the profile is being presented. The full list of information from the sample preset database is being shown. Query found under appendix A13 has been used to generate this data. This is the end of the Statistics flow.

---

### 3.0

The final published build for the project failed to meet many of the target goals of the previous V1.03 build. A lot of the constraints this time fall as part of developer burnout, with it being very difficult to concentrate on meeting the required goals as well as making more frequent errors when coding.

This, however, did not stop the final submitted product from meeting minimum requirements of fully reading from the sample preset database and the creation of an additional screen that displays data. Unfortunately, this also came at the cost of

testing of the software being extremely limited, so the amount of errors being caught was extremely low. This will not be the last build of the product however, and future versions may appear as part of a github project.

---

## Appendix A

### Additional resources

---

Contained in *Appendix A* are additional resources in order by which they are referenced in the documentation. Most of what is contained are diagrams and tables, followed by a brief explanation to ensure full clarity of what is being displayed. All images will be additionally attached within the zipped folder in order to see smaller details not visible within the report.

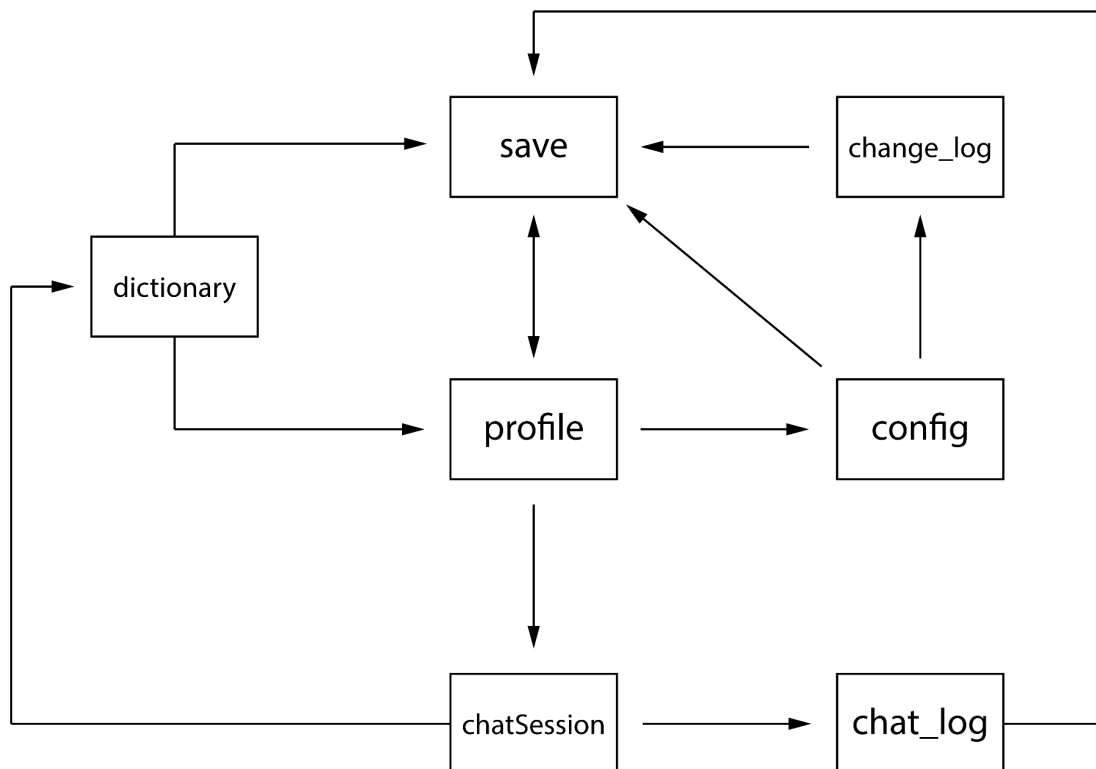


Figure A1. Flow diagram of MedusaUI program.

The flow diagram displayed above demonstrates how entities would communicate with one another in a functional state. Comments were made on the circular structure of the respective ER Diagram found as *Figure A2*.

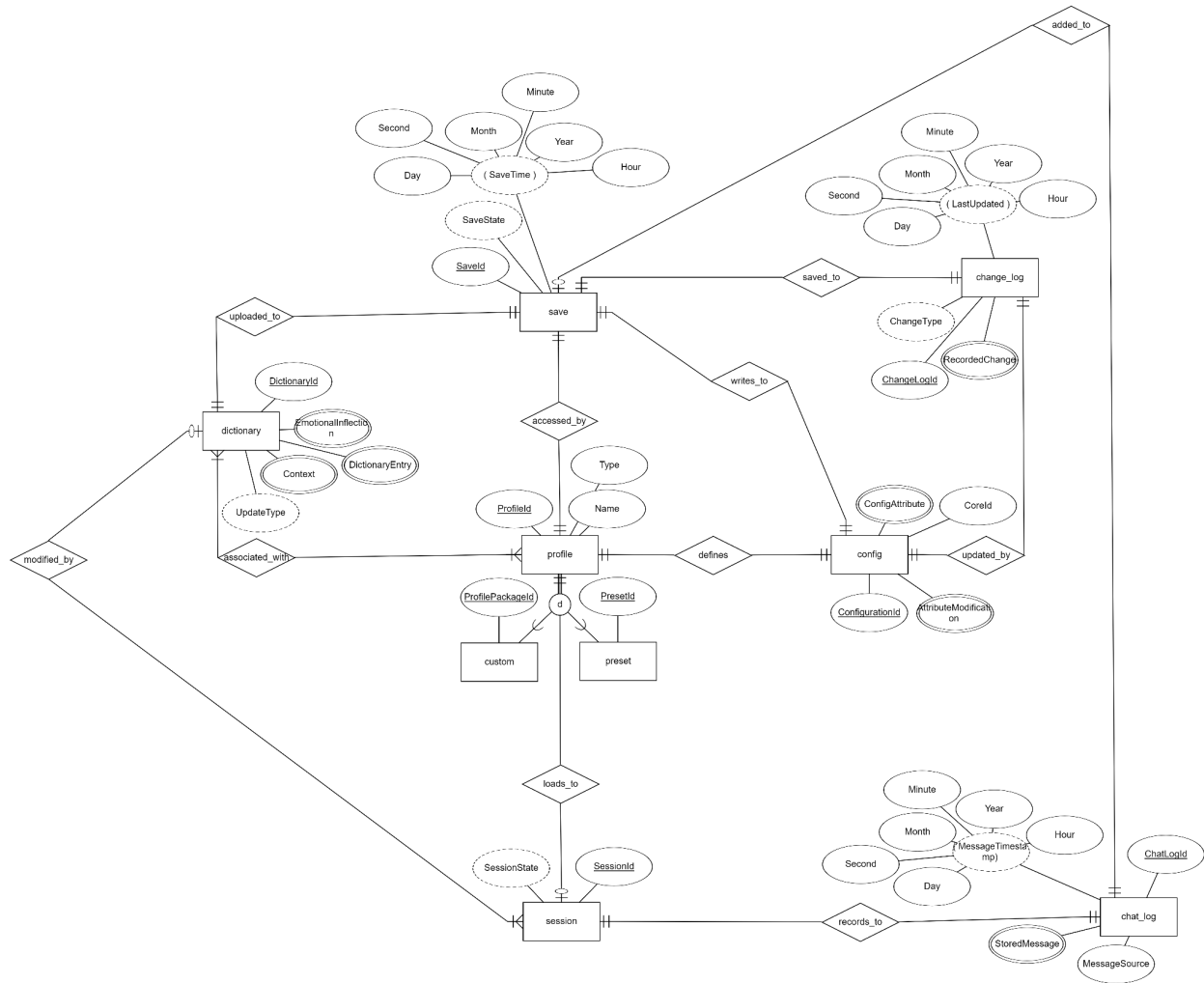


Figure A2. MedusaUI entity-relation diagram.

The ER diagram for the MedusaUI program had a variety of changes made to it, mostly in the form of renaming of entities/relations. The most major change is the abandoning of the specialisation relation between preset and custom profiles, as functionally no advantage was gained from it.

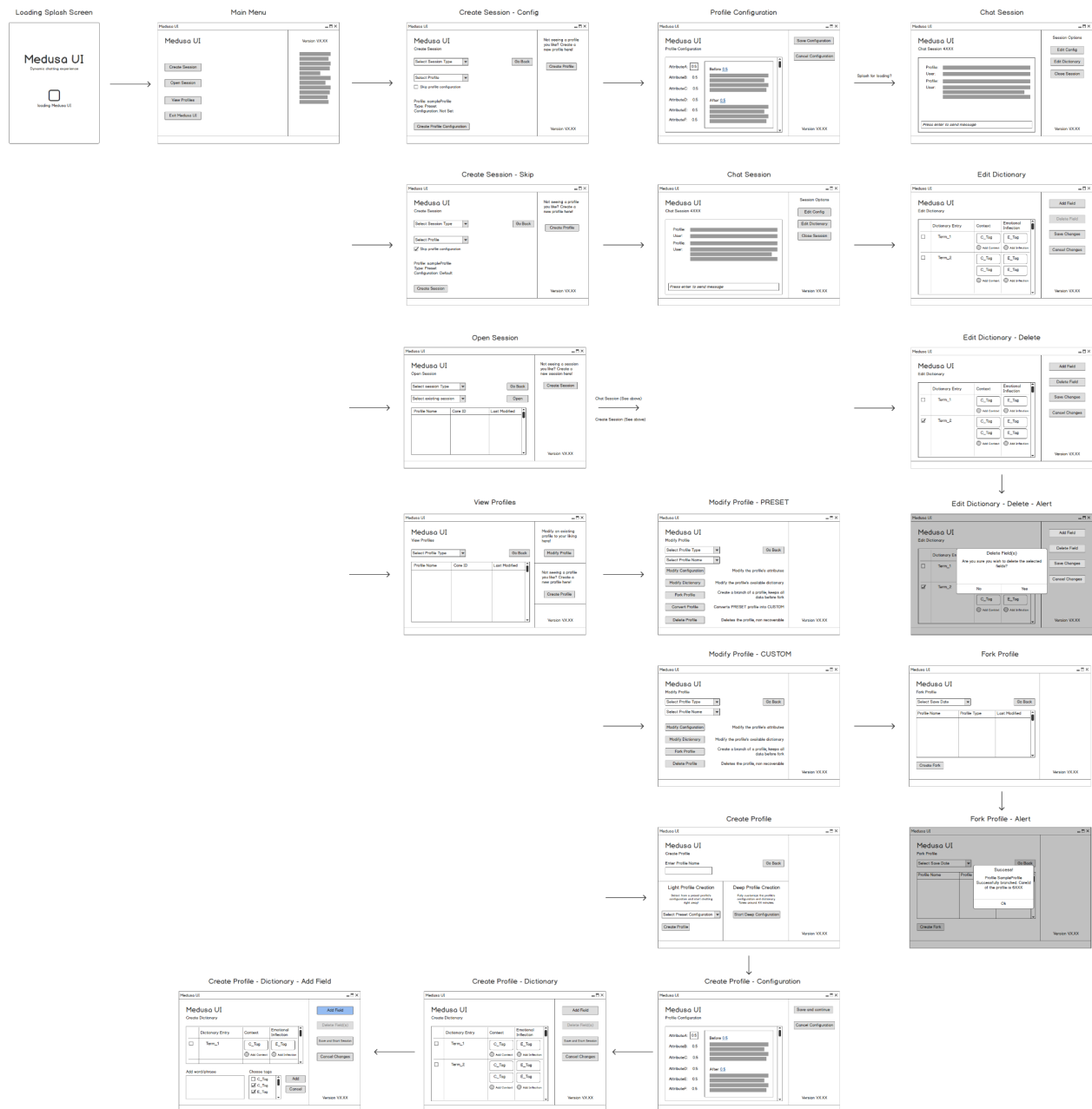


Figure A3. Mockup of the MedusaUI program interface.

The mockup was designed throughout the first week of the program's development and features the overall layout flows of the program. Future versions will feature more modifications to the interface that will not be reflected in the mockup.



```
SELECT Name, Type, config.CoreId AS cores,  
change_log.LastUpdated AS lastUpd FROM profile  
INNER JOIN defines ON defines.ProfileId = profile.ProfileId  
INNER JOIN config ON config.ConfigurationId =  
defines.ConfigurationId  
INNER JOIN updated_by ON updated_by.ConfigurationId =  
config.ConfigurationId  
INNER JOIN change_log ON change_log.ChangeLogId =  
updated_by.ChangeLogId;
```

Figure A4. SQL query to load in Name, Type, CoreId, and LastUpdated fields.

This is a custom query designed to import the 4 base parameters required for most of the program to function. Since all of the required fields were tightly clustered around each other, this allowed for easy access to the required fields through connected tables.

```
SELECT ConfigAttribute FROM config  
INNER JOIN defines ON defines.ConfigurationId =  
config.ConfigurationId  
INNER JOIN profile ON profile.ProfileId = defines.ProfileId  
WHERE profile.Name = [NAME]
```

Figure A5. SQL query to retrieve the configuration associated with a profile name.

This is a custom query designed to import a list of configuration modifiers for a profile. The current implementation uses a very basic link of profile name to determine the configuration, however a more accurate way to determine the exact profile configuration would be to select a specific core ID as well, since a forked profile has the potential to return multiple configurations. To use the query manually in DB Browser, replace [NAME] with an existing profile name.

```

SELECT DictionaryEntry, Context, EmotionalInflection FROM
dictionary
INNER JOIN associated_with ON associated_with.DictionaryId =
dictionary.DictionaryId
INNER JOIN profile ON profile.ProfileId =
associated_with.ProfileId WHERE profile.Name = [NAME]

```

Figure A6. SQL query to retrieve the dictionary associated with a profile name.

This is a custom query designed to import a dictionary associated with a profile. The current implementation uses a very basic link of profile name to determine the dictionary, however a more accurate way to determine the exact profile dictionary would be to select a specific core ID or profileId as well, since a forked profile has the potential to return multiple dictionaries. To use the query manually in DB Browser, replace [NAME] with an existing profile name.

```

SELECT dictionary.DictionaryEntry,
MAX(length(EmotionalInflection)-length(replace(EmotionalInflection, ',', ''))+1+(length(Context)-length(replace(Context, ',', ''))+1) ) TagCount
FROM dictionary
INNER JOIN associated_with on associated_with.DictionaryId =
dictionary.DictionaryId
INNER JOIN profile ON profile.ProfileId =
associated_with.ProfileId
INNER JOIN defines ON defines.ProfileId = profile.ProfileId
INNER JOIN config ON config.ConfigurationId =
defines.ConfigurationId
WHERE profile.Name = [NAME] AND config.CoreId = [COREID]
GROUP BY dictionary.DictionaryId, dictionary.DictionaryEntry
ORDER BY TagCount DESC
LIMIT 1;

```

Figure A7. SQL query to retrieve the word with the highest tag count from a dictionary associated with a profile name and coreID.

This is a custom query that returns the top result of a custom table that counts and sums up the amount of context and emotional inflection tags associated with a dictionary entry. The search is specific by profile name and core ID to ensure that a forked profile will not return the entries associated with the base profile. [NAME] and [COREID] are replaced by the specific name and coreID of a profile.

```
SELECT change_log.ChangeLogId, change_log.RecordedChange,
change_log.ChangeType, change_log.LastUpdated FROM change_log
INNER JOIN saved_to ON saved_to.ChangeLogId =
change_log.ChangeLogId
INNER JOIN save ON save.SaveId = saved_to.SaveId
INNER JOIN accessed_by ON accessed_by.SaveId = save.SaveId
INNER JOIN profile ON profile.ProfileId = accessed_by.ProfileId
INNER JOIN writes_to ON writes_to.SaveId = save.SaveId
INNER JOIN config ON config.ConfigurationId =
writes_to.ConfigurationId
WHERE profile.Name = [NAME] AND config.CoreId = [COREID] AND
change_log.RecordedChange = "Session Opened"
```

Figure A8. SQL query to retrieve the time at which a chat session was opened.

This is a simple custom query that combined the data of multiple tables to help retrieve the row where the target profile's session was recorded to have opened. 3 Fields were used to target the specific row, with profile name and coreID being the generic 2, and the additional RecordedChange field to search for the "Session Opened" entry. [NAME] and [COREID] are replaced by the specific name and coreID of a profile.

```
SELECT change_log.ChangeLogId, change_log.RecordedChange,
change_log.ChangeType, change_log.LastUpdated FROM change_log
INNER JOIN saved_to ON saved_to.ChangeLogId =
change_log.ChangeLogId
INNER JOIN save ON save.SaveId = saved_to.SaveId
INNER JOIN accessed_by ON accessed_by.SaveId = save.SaveId
INNER JOIN profile ON profile.ProfileId = accessed_by.ProfileId
INNER JOIN writes_to ON writes_to.SaveId = save.SaveId
```

```
INNER JOIN config ON config.ConfigurationId =
writes_to.ConfigurationId
WHERE profile.Name = [NAME] AND config.CoreId = [COREID] AND
change_log.RecordedChange = "Session Closed"
```

Figure A9. SQL query to retrieve the time at which a chat session was closed.

This is a simple custom query that combined the data of multiple tables to help retrieve the row where the target profile's session was recorded to have closed. 3 Fields were used to target the specific row, with profile name and coreID being the generic 2, and the additional RecordedChange field to search for the "Session Closed" entry. [NAME] and [COREID] are replaced by the specific name and coreID of a profile.

```
SELECT chatSession.SessionId from chatSession
INNER JOIN records_to ON records_to.SessionId =
chatSession.SessionId
INNER JOIN chat_log ON chat_log.ChatLogId =
records_to.ChatLogId
INNER JOIN added_to ON added_to.ChatLogId = chat_log.ChatLogId
INNER JOIN save ON save.SaveId = added_to.SaveId
INNER JOIN writes_to ON writes_to.SaveId = save.SaveId
INNER JOIN saved_to ON saved_to.SaveId = save.SaveId
INNER JOIN change_log ON change_log.ChangeLogId =
saved_to.ChangeLogId
INNER JOIN config ON config.ConfigurationId =
writes_to.ConfigurationId
INNER JOIN defines ON defines.ConfigurationId =
config.ConfigurationId
INNER JOIN profile ON profile.ProfileId = defines.ProfileId
WHERE profile.Name = [NAME] AND config.CoreId = [COREID] AND
change_log.RecordedChange = "Session Closed"
```

Figure A10. Modified query to retrieve the chat session of the successfully closed session.

This query was extremely specific, and that was to retrieve the specific sessionID of the closed session record associated with the specific profile name and coreID. Since sessionID was

found in a deeper layer, the query had to be fully restructured based on similar logic to *Figure A9*. Associating the session with the open time would work, however since it was used in par with finding the time length of a chat session, it was decided to determine the sessionID by the closing record. [NAME] and [COREID] are replaced by the specific name and coreID of a profile.

```
SELECT chatSession.SessionId from chatSession
INNER JOIN records_to ON records_to.SessionId =
chatSession.SessionId
INNER JOIN chat_log ON chat_log.ChatLogId =
records_to.ChatLogId
INNER JOIN added_to ON added_to.ChatLogId = chat_log.ChatLogId
INNER JOIN save ON save.SaveId = added_to.SaveId
INNER JOIN writes_to ON writes_to.SaveId = save.SaveId
INNER JOIN saved_to ON saved_to.SaveId = save.SaveId
INNER JOIN change_log ON change_log.ChangeLogId =
saved_to.ChangeLogId
INNER JOIN config ON config.ConfigurationId =
writes_to.ConfigurationId
INNER JOIN defines ON defines.ConfigurationId =
config.ConfigurationId
INNER JOIN profile ON profile.ProfileId = defines.ProfileId
WHERE profile.Name = [NAME] AND config.CoreId = [COREID] AND
chatSession.SessionState = "CLOSED"
```

Figure A11. Query used to find the sessionID of a successfully closed chat session.

This is the most specific query, as it searches for only the successfully closed session record associated with a profile. [NAME] and [COREID] are replaced by the specific name and coreID of a profile.

```
SELECT change_log.ChangeLogId, change_log.RecordedChange,
change_log.ChangeType, change_log.LastUpdated FROM change_log
INNER JOIN saved_to ON saved_to.ChangeLogId =
change_log.ChangeLogId
INNER JOIN save ON save.SaveId = saved_to.SaveId
INNER JOIN accessed_by ON accessed_by.SaveId = save.SaveId
```

```
INNER JOIN profile ON profile.ProfileId = accessed_by.ProfileId
INNER JOIN writes_to ON writes_to.SaveId = save.SaveId
INNER JOIN config ON config.ConfigurationId =
writes_to.ConfigurationId
WHERE profile.Name = [NAME] AND config.CoreId = [COREID]
```

Figure A12. Query used to retrieve change log.

This query is very basic, it just retrieves the associated change log with a profile by profile name and coreID. [NAME] and [COREID] are replaced by the specific name and coreID of a profile.

```
SELECT chat_log.ChatLogId, chat_log.StoredMessage,
chat_log.MessageSource, chat_log.MessageTimestamp FROM chat_log
INNER JOIN added_to ON added_to.ChatLogId = chat_log.ChatLogId
INNER JOIN save ON save.SaveId = added_to.SaveId
INNER JOIN accessed_by ON accessed_by.SaveId = save.SaveId
INNER JOIN profile ON profile.ProfileId = accessed_by.ProfileId
INNER JOIN defines ON defines.ProfileId = profile.ProfileId
INNER JOIN config ON config.ConfigurationId =
defines.ConfigurationId
WHERE profile.Name = [NAME] AND config.CoreId = [COREID]
```

Figure A12. Query used to retrieve chat log.

This query is very basic, it just retrieves the associated chat log with a profile by profile name and coreID. [NAME] and [COREID] are replaced by the specific name and coreID of a profile.