mzDBWizard User Guide

Introduction

mzDBWizard was born in order to alter the labor intensive workflow that is followed when .raw files need to be converted and uploaded. It is a simple tool whose purpose is relieve the end user from the burden that chaperons the manual manipulation of .raw files. In order to achieve this, the application was built with a daemon-centric logic in mind requiring minimum interaction from the part of the user.

It works by automatically detecting file creations within a directory of the local file system. As long as a new file is detected, appropriate actions take place depending on the file type. If the latter one corresponds to a .raw file, a conversion mechanism is activated leading to the creation of an .mzdb file relying on the well established, raw2mzDB converter. On the other hand, when an .mzdb file is created within the monitored folder, **mzDBWizard** uploads it by using the existing functionality and infrastructure of the server.

Installation

System Requirements

The requirements that derive from software's local and network activity can be summarized into the three following prerequisites:

1. Java SE JRE of Version 7 or newer installed
2. An running Proline Server
3. A working copy of **raw2mzDB** converter

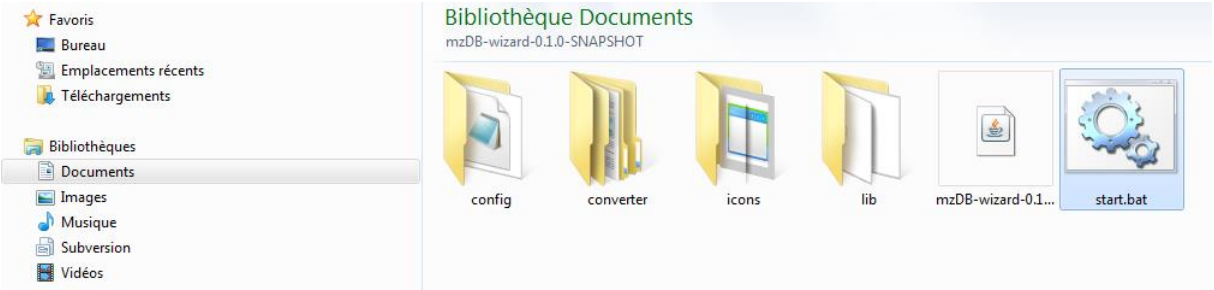
Installation Using Archive Files

Given that all the aforementioned requirements are met, the installation of **mzDBWizard** software is a straightforward procedure as it only demands the respective **mzDBWizard** archive to be locally decompressed. From that moment and as long as all three requirements are respected the software is ready for launch and use.

Getting Started

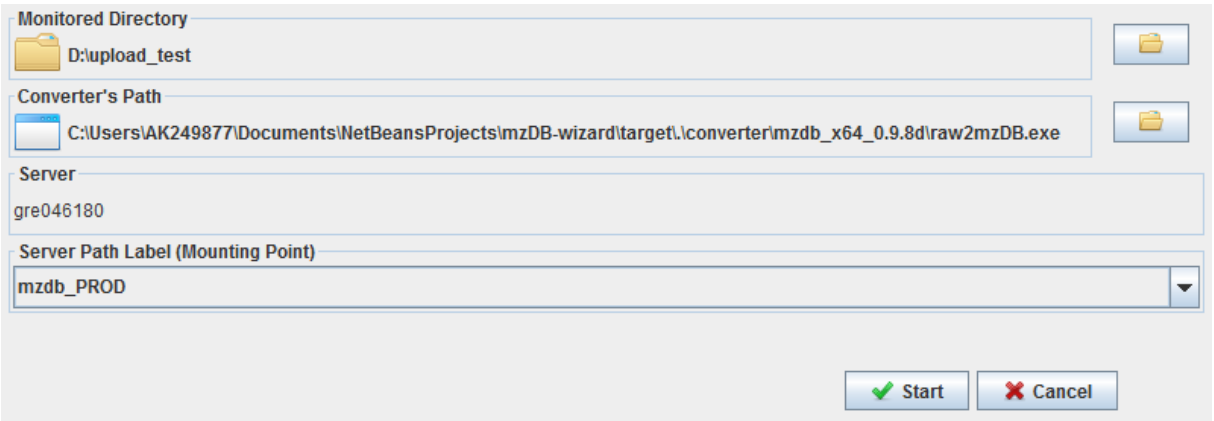
Launching mzDBWizard

mzDBWizard can be launched executing the file start.bat which can be found inside the decompressed root folder as shown in the following screenshot.



Opening Dialog

Start.bat's execution is always followed by an opening dialog that serves as a configurator for the current execution of **mzDBWizard**. The dialog encapsulates input fields for all the information that is required by the program. As seen on the respective screenshot, when the program is initiated, the user has to dictate the folder to be monitored, the **mzDBWizard** copy, the server's address or alias and the respective mounting point. In an effort to minimize the required workload, the filled in information is saved, eliminating the need to fill this information every time unless a modification is needed.

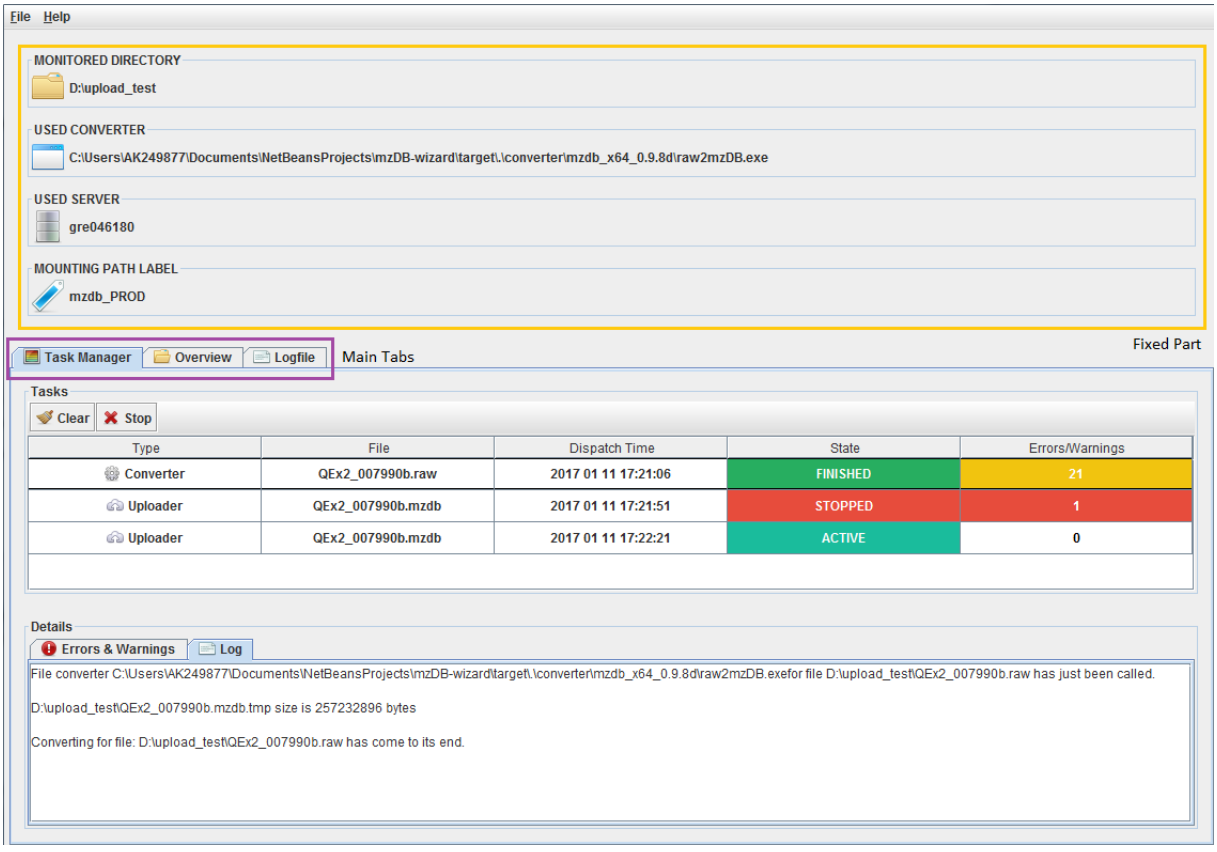


Main User Interface

mzDBWizard has a built-in graphical user interface. The latter one, inextricably linked to the minimization of the user's workload serves a range of purposes, including among others:

- Conversion/upload tasks management
- Conversion/upload tasks extensive monitoring
- Configuration
- Debugging

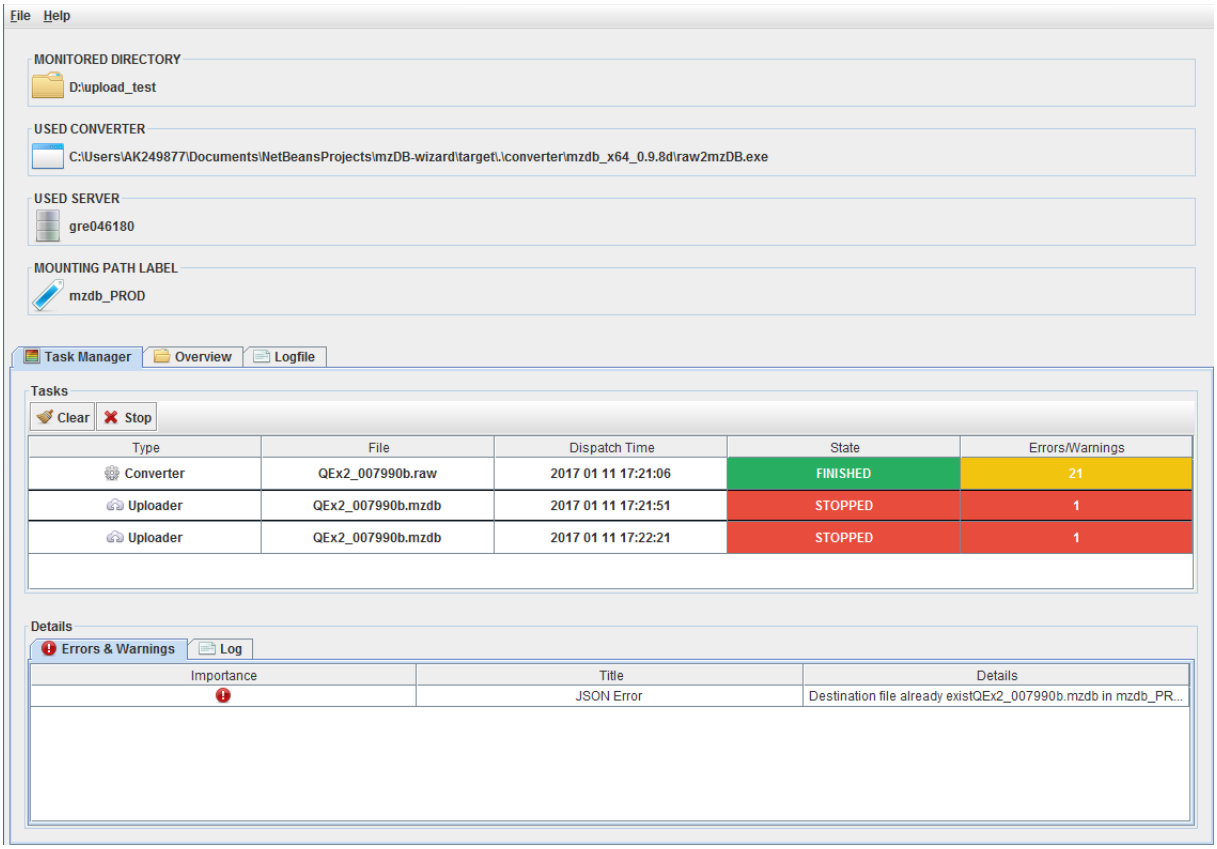
Following this generic wireframe, the interface was organized into three main tabs, plus one fixed part that provides a quick overview of the current execution's attributes (Monitored directory, Converter in use, Server etc.).



Task Manager Tab

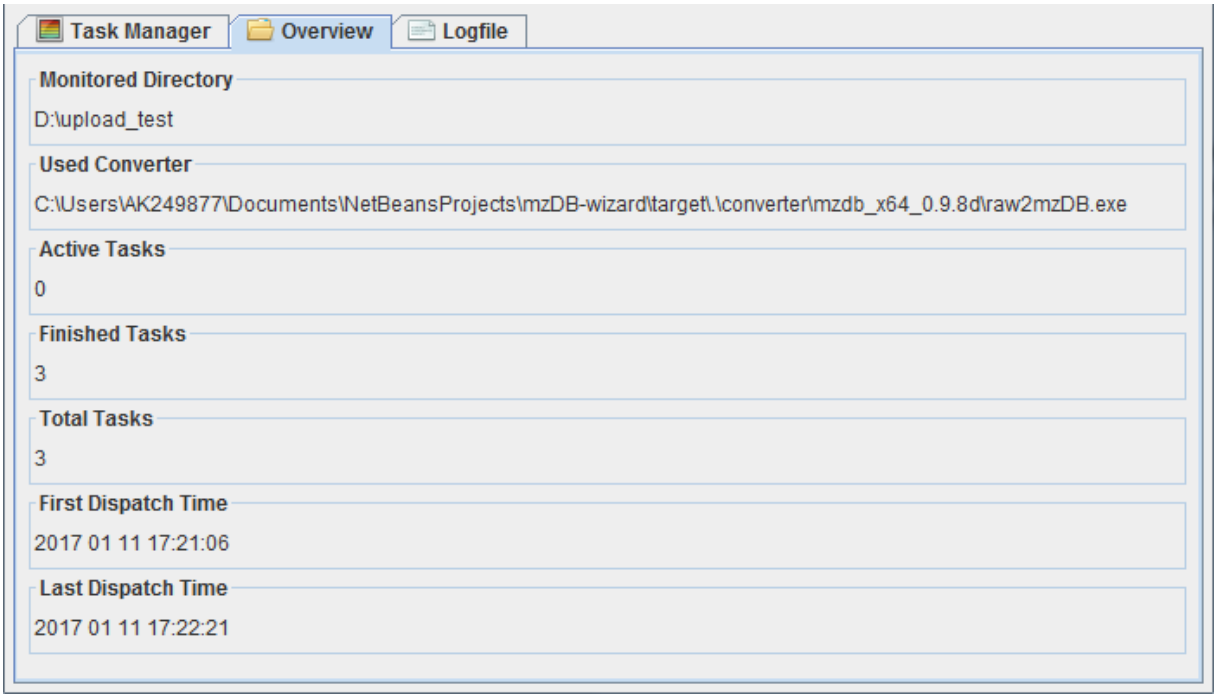
The task manager dedicated tab, which can be seen at the previous screenshot is the cornerstone of the interface. Its main virtue is the fact that it provides very detailed information about all the launched tasks in an easily digestible way, using different elements (colors codes, image icons). Furthermore, apart from the provided situation awareness, it also provides the user with the means to manage tasks in the form of buttons and pop-up menus. In this context, the user has the ability to stop, restart or clear a task. It must be noted that a task that has entered its critical part cannot be

killed. A critical point can be seen as a “Rubicon” in the process which usually has no return. An example of a critical point in the process of uploading an .mzdb file is the establishment of the required connection with the server. Once the connection is active, killing the connection would lead to corrupted files at the server’s side. As it is also seen in following screenshot, task manager also consists of two sub-tabs, one dedicated to logs and one that serves as a summary of all the encountered errors and warnings. Both sub-tabs regard only the selected task, highlighted with a black border.



Overview Tab

As its name denotes, overview serves as a general overview focusing on the launched tasks on the contrary with the fixed part of the graphical user interface, as defined earlier, that although serves a very similar purpose, focuses on the current execution's attributes.

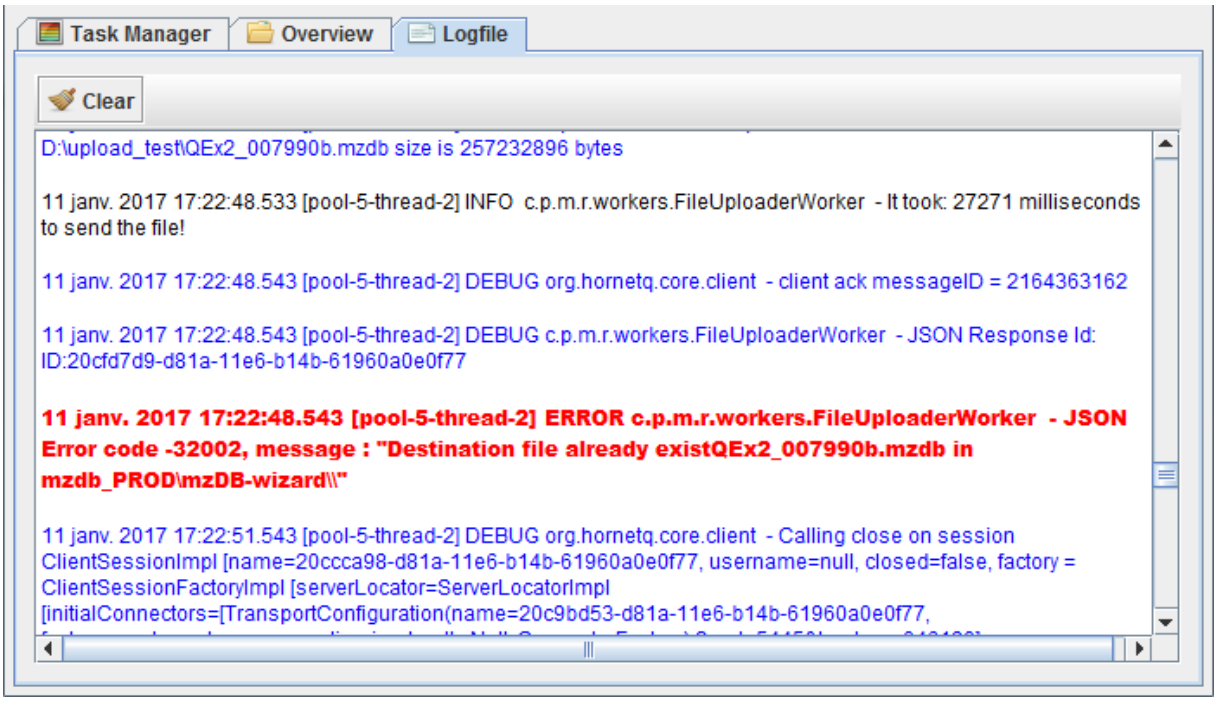


Logfile Tab

The logfile tab can be compared with a centralized logfile regarding the launched tasks as the well as the application itself. As we can see, the logs themselves are organized using different color codes:

- **Bold Red**, for error messages
- Blue, for debugging messages
- Black, for information messages

Although it is available for the user, it is highly unlike that the end user will use it as it was mainly built for debugging reasons. In case that he wants to access logs, the relevant task manager subtab can be used. An advantage for the latter one is that it corresponds to a decluttered version.



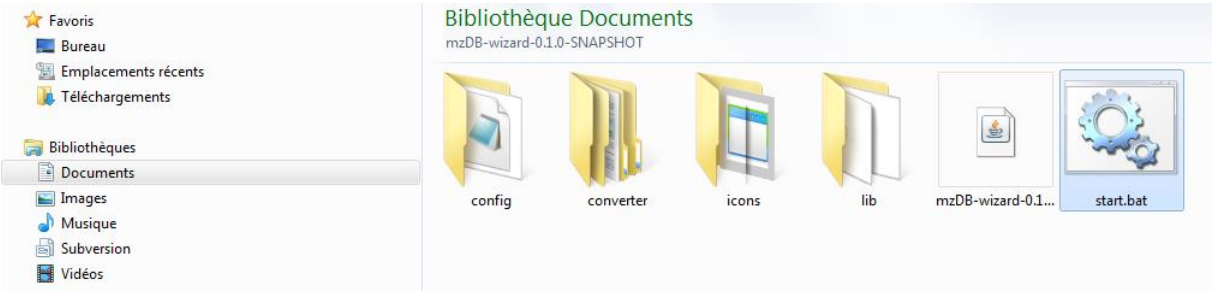
Use Case Example

In order to familiarize ourselves with **mzDBWizard** we will proceed with a Use Case Example, demonstrating the basic functionality of the software. The goal is to launch, configure and then process a user folder that contains both .raw and .mzdb files. For the sake of completeness we will also deliberately provoke an error so that the user becomes familiar with error handling. The example's workflow is organized into eight (8) steps as follows:

1. Launch **mzDBWizard**
2. Initiate the application using opening dialog
3. Modify preferences
4. Trigger a detection
5. Visualize progress
6. Provoke/handle an abnormal situation

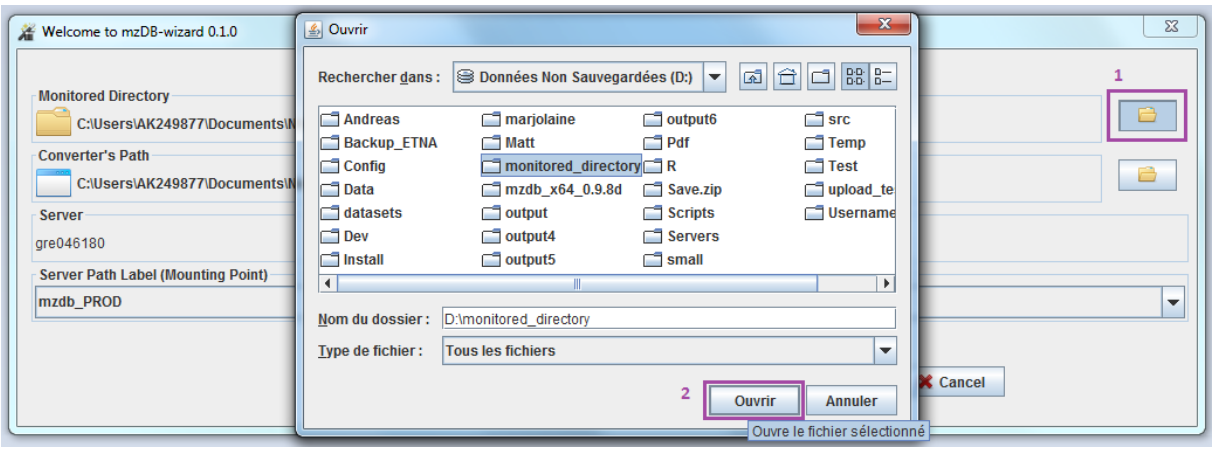
1. Launch mzDBWizard

As mentioned before, in order to launch **mzDBWizard** it suffices to execute the file start.bat, located inside **mzDBWizard** root directory.



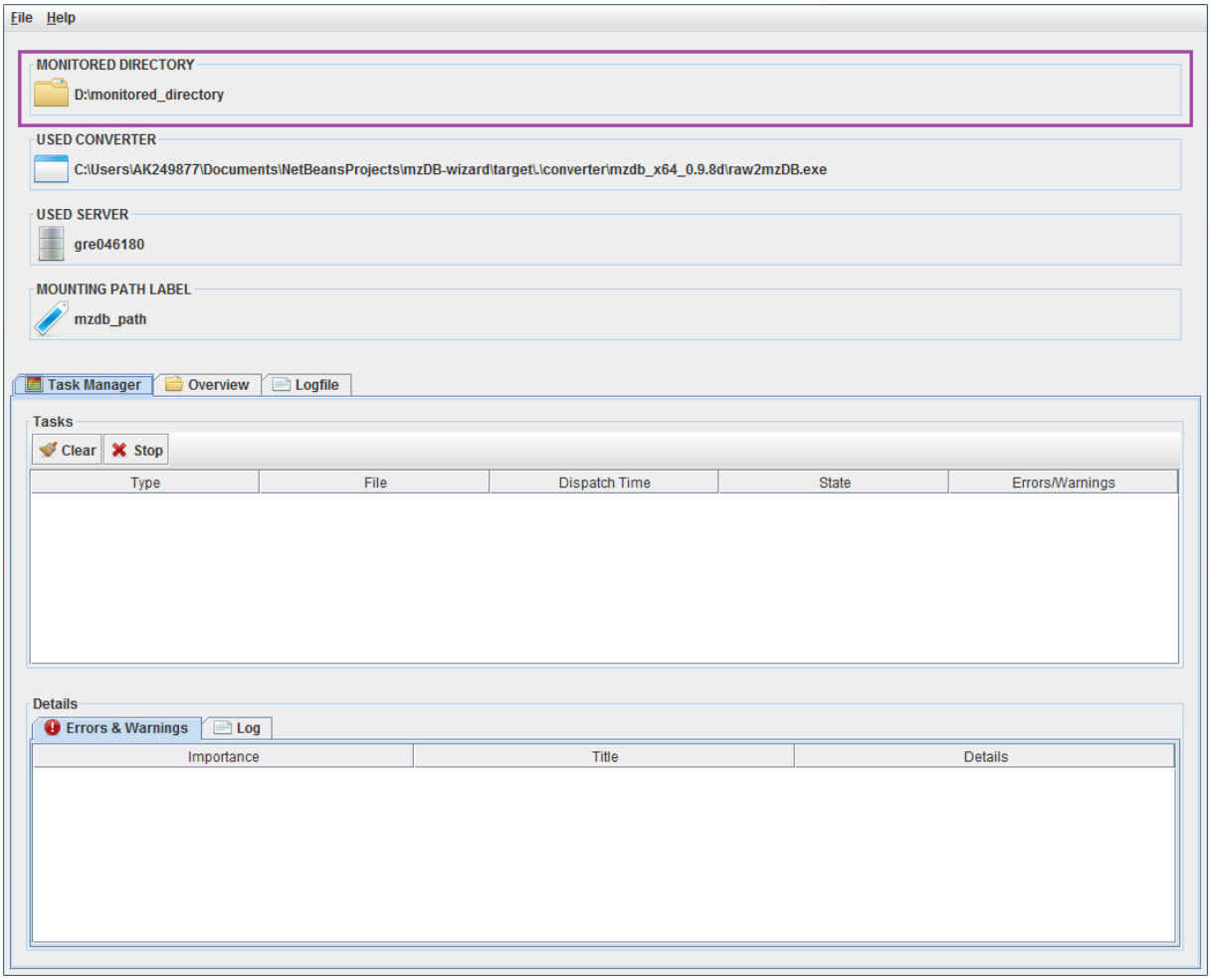
2. Initiate the application using the opening dialog

Once start.bat file is executed, the opening dialog of **mzDBWizard** will appear. Based on the assumption that this is the first execution of the application we will have to verify or modify the respective form accordingly starting from the selection of the directory to be monitored. We select the on purpose created “monitored_directory” and press ok as shown in the following screenshot.



The next item to our initial configuration is converter that will be used for this execution cycle. As we can see, if not changed, the selected “Converter’s Path” points to the converter executable file that is shipped/packed with **mzDBWizard**. In case that a different version is desired due to the special nature of a file in question, another converter executable can be selected following the same logic with the selection of the “Monitored Directory”. The last two items to check on the opening dialog are the server’s address and the respective mounting point. For the purposes of this tutorial we will leave them as they are, as they correspond to the laboratory’s server. Finishing the configuration we press ok, event that automatically starts the monitoring activity and shows the main dialog of the

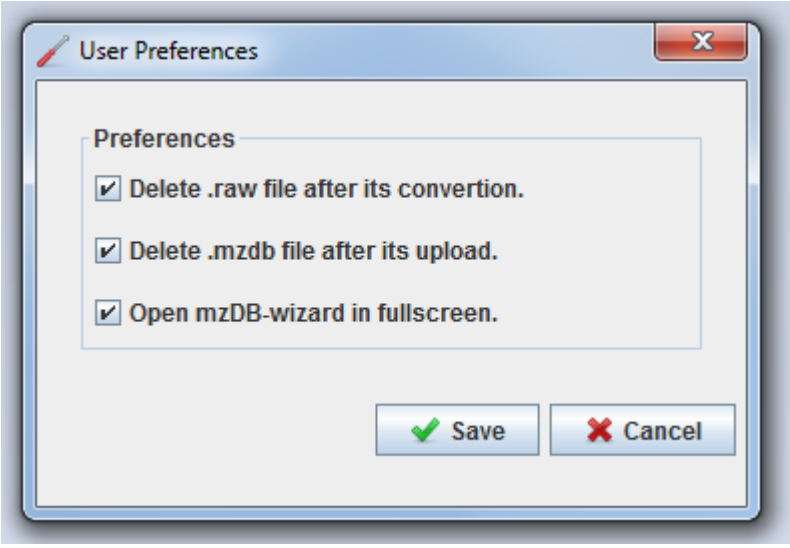
application. Note that as mentioned before, we can validate the configuration that just took place by looking to the fixed part at the top of the dialog.



We observe that, “D:\monitored_directory” is indeed the folder that is currently under the diligent “eye” of the application. Since the newly created directory is currently empty, no task exist yet.

3. Modify Preferences

Before triggering our first file detection, it is appropriate to have a first contact with the preferences dialog within the application. The latter can be accessed using the menu-bar at the top and gives us access to a short list of options regarding the behavior of the application. For the sake of completeness we must state that more options do exist in a respective configuration file. As their more technical nature surpasses the boundaries of this Use Case Example so we will refrain from referring to them for now. The two most significant items on the dialog are the two ones that regard the deletion of a file after an action has taken place. The features were introduced as an answer to the awareness that most of the time the application will be executed on shared laboratory computers with limited local space for which different users compete. At this point it is important to add to the self-explanatory labels that a deletion takes place only when a task was successfully finished. In cases of critical events the deletion is aborted preventing any possible loss of data.

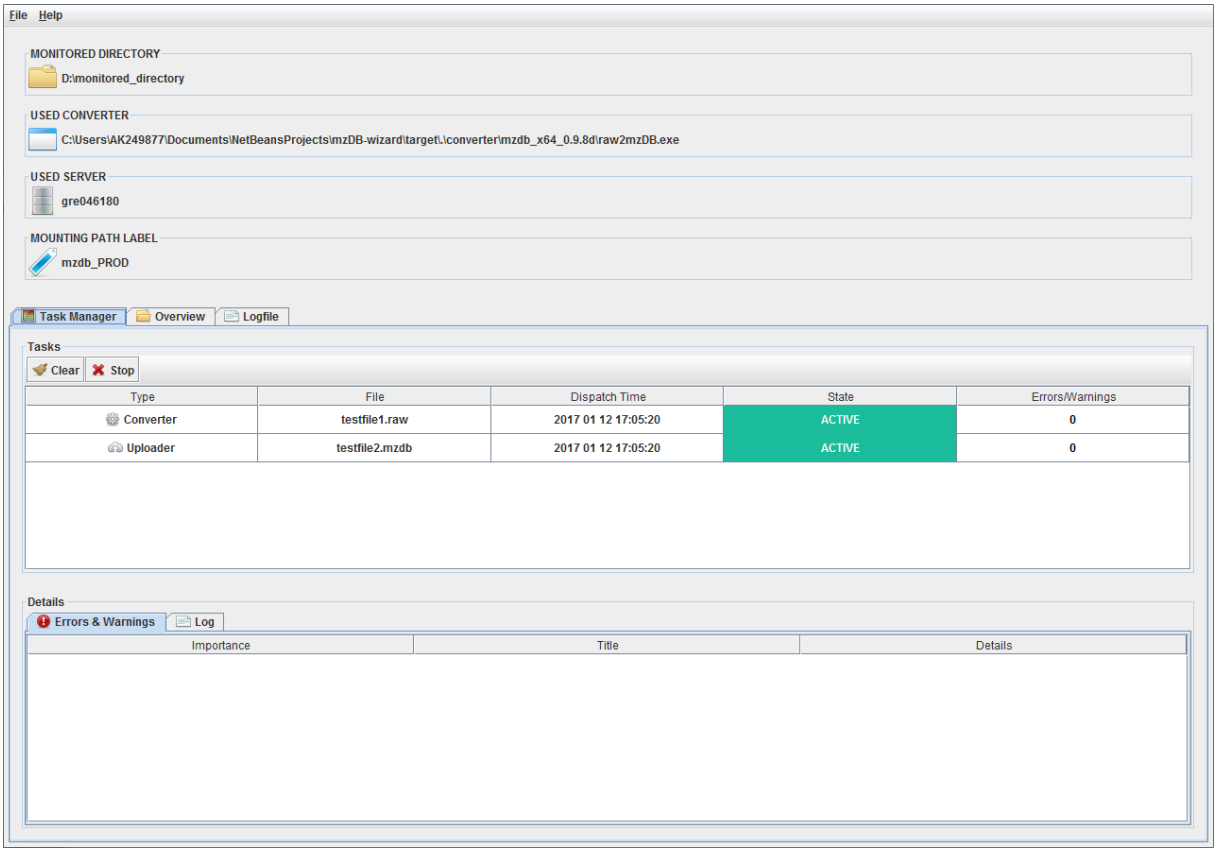


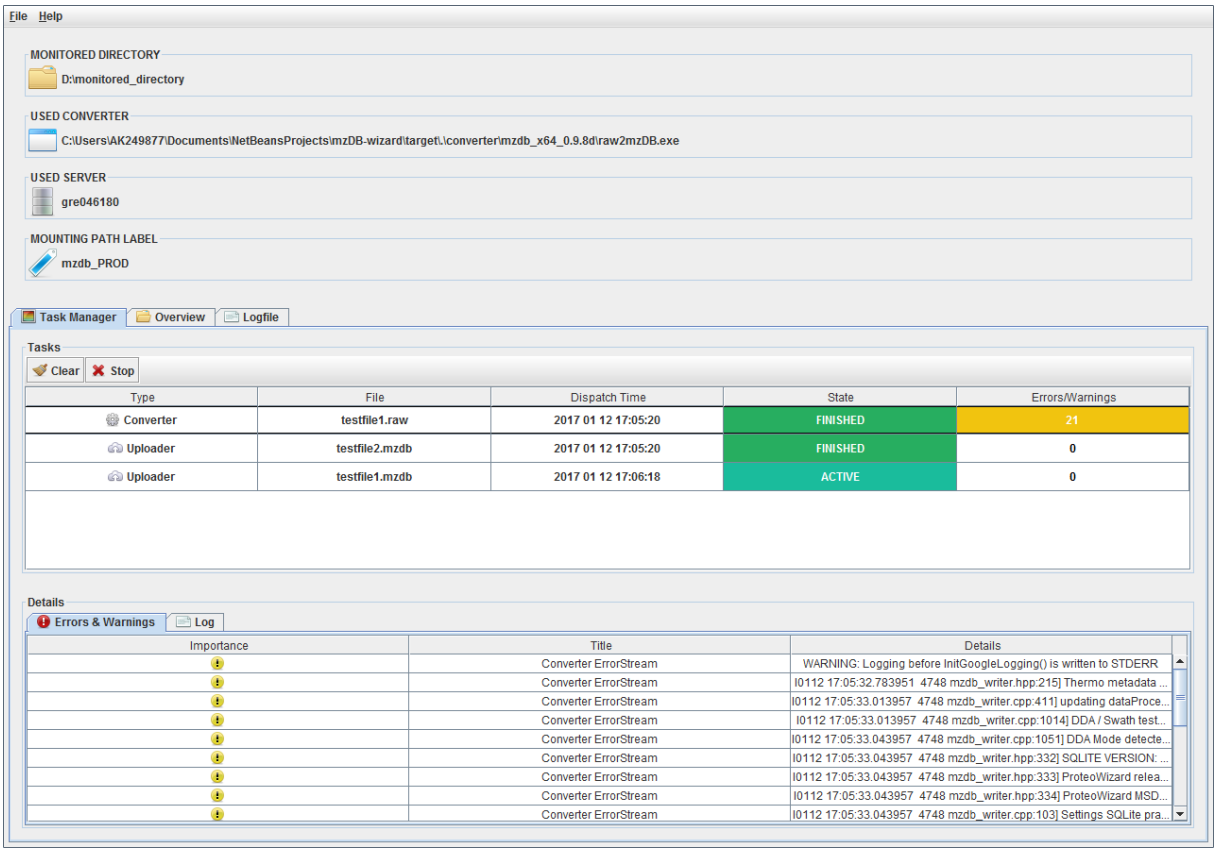
4. Trigger a detection

As previously explained, from the moment the main dialog appears, the selected directory will be monitored in a recursive way. The latter feature can be exploited in order to have a more organized destination directory. It is thus recommended that you firstly create a new folder, inside which you will deposit the files that are of your interest and that would like to convert and upload. At the destination, files will be saved respecting your structure, regardless of its depth, creating any not already existing participating directories. One good practice could thus be to create a folder named after a name or username and then create a number of subfolders named after an attribute like the project name. For the purposes of our example we consider as suffice to create one root folder named “username” which contains only one subfolder named “project-x” consisting of two files, “testfile1.raw” and “testfile2.mzdb”. In order to trigger the detection it is sufficient to just copy “username” folder into “D:\monitored_directory”. This will dispatch the required procedures that can be seen at the task manager at the following screenshot.

5. Visualize Progress

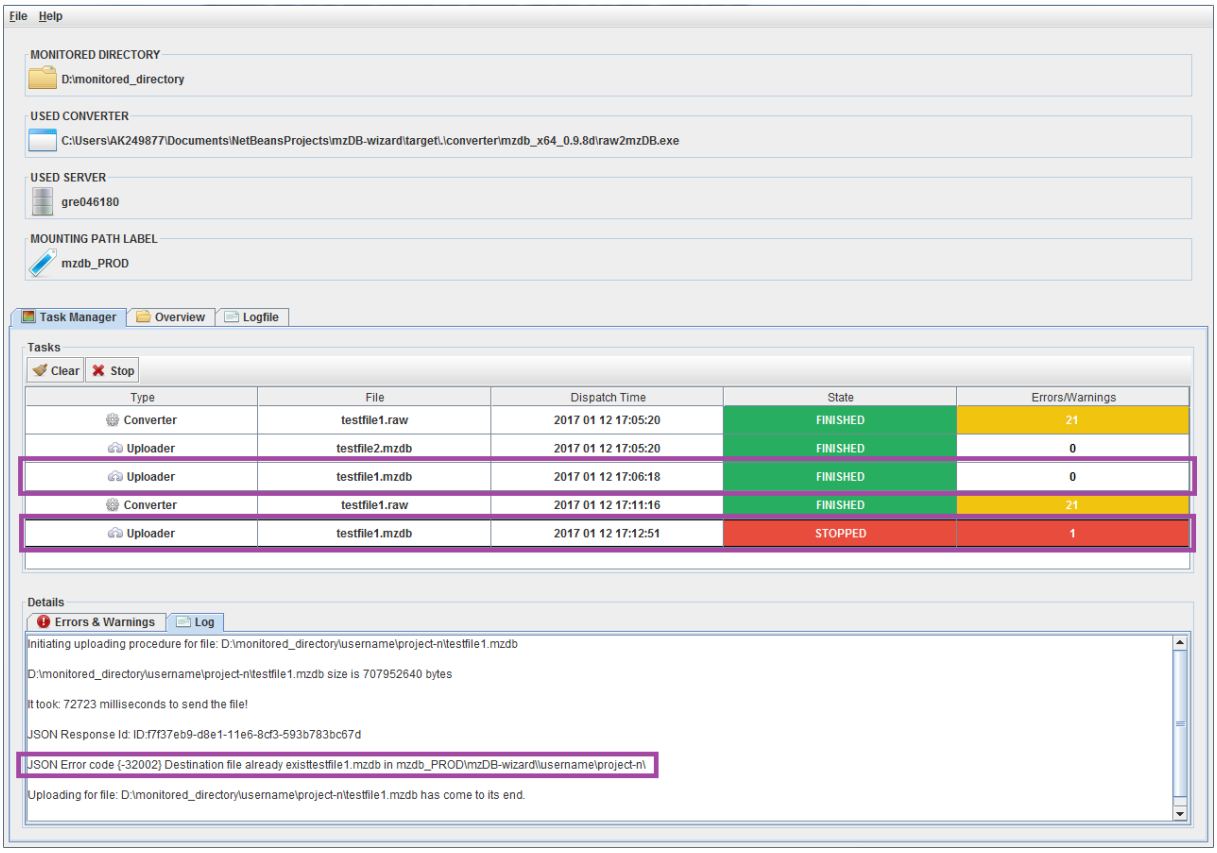
As we can see, both files are successfully detected and are processed by appropriate procedures. At this point it is important to remember that we can either drop a .raw file, expecting it to be converted and then uploaded or directly an .mzdb file. The latter one does not need conversion, it will thus be directly uploaded. The following screenshot is taken a few seconds later showing two finished tasks plus one active one. Being observant we can distinguish that the conversion task finished with twenty one (21) warnings but no critical errors leading to the creation of "testfile1.mzdb" which triggered one more upload task shown as active at the moment the screenshot was taken.





6. Provoke/handle an abnormal situation

Once more, for the sake of completeness we will derive from the normal workflow in order to replicate an abnormal situation, or what is considered as a critical error within our application. One simple error that can occur is a duplicate file in destination. It occurs when an upload takes for a file that already exists in destination at the exact same path. As we can see in the relevant screenshot this will provoke a critical error. During the latter one, the state of the task in question will initially change to not-responding and will then be automatically be killed by the application itself. Moreover, given the fact that it failed to upload, even if in preferences, “Delete .mzdb file after upload” is selected file will not be deleted. Regarding handling, the application follows a try-once-more approach according to which it will try to restart once more a critically failed task. In case of a second failure the task is not restarted automatically demanding the explicit restarting order from the user. The order is given through by right clicking on the killed task and selecting the restarting option.


File Help
MONITORED DIRECTORY
D:\monitored_directory
USED CONVERTER
C:\Users\AK249877\Documents\NetBeansProjects\mzDB-wizard\target\.\converter\mzdb_x64_0.9.8d\raw2mzDB.exe
USED SERVER
gre046180
MOUNTING PATH LABEL
mzdb_PROD
Task Manager
Overview
Logfile
Tasks
Clear
Stop
Type
File
Dispatch Time
State
Errors/Warnings
Converter
testfile1.raw
2017 01 12 17:05:20
FINISHED
21
Uploader
testfile2.mzdb
2017 01 12 17:05:20
FINISHED
0
Uploader
testfile1.mzdb
2017 01 12 17:06:18
FINISHED
0
Converter
testfile1.raw
2017 01 12 17:11:16
FINISHED
21
Uploader
testfile1.mzdb
2017 01 12 17:12:51
STOPPED
1
Details
Errors & Warnings
Log
Initiating uploading procedure for file: D:\monitored_directory\username\project-n\testfile1.mzdb
D:\monitored_directory\username\project-n\testfile1.mzdb size is 707952640 bytes
It took: 72723 milliseconds to send the file!
JSON Response Id: ID:f7f37eb9-d8e1-11e6-8cf3-593b783bc67d
JSON Error code {-32002} Destination file already existtestfile1.mzdb in mzdb_PROD\mzDB-wizard\\username\project-n\
Uploading for file: D:\monitored_directory\username\project-n\testfile1.mzdb has come to its end.