Doxygen Tutorial

Jafar Al-Kofahi

Every file must have a File Header documentation section as described in the Coding Standard document, the file headers must be documented using the following format:

```
/**
*     @file util.h
*     @brief this header file will contain all required
*     definitions and basic utilities functions.
*
*     @author     Dr.Zhao Zhang
*
*     @date 2/28/2009
*/
```

The ***@file*** must be used to declare the file name, and then you must use ***@brief*** to give a brief description of your class, it will take the first statement after it as a description (till the fist "."). Everything else after that will be taken as a detailed description of the class except when other Doxygen commands are detected. Use the ***@author*** command to specify the author name of this class. And at the end add the ***@date*** with the date the file created.

All functions must have a Function Header documentation section as described in the Coding Standard document, and you must use the following format for that:

```
/**
*     This method will be used to print a single character to the lcd.
*     @author Dr.Zhao Zhang
*     @param chrData    The character to print
*     @date 2/28/2009
*/
```

The first sentence is a brief description of the function, then the author of the function (not necessary the same as the author of the file), then list all of your parameters using ***@param*** (every parameter must have @param infront of it), and in the end add the date in which this function was created.

What you should document in the headers for both files and functions is the goal of having them, and what is their purpose, note that headers should not document any technical details at all. For the files, there is no technical documentation since this will be achieved by documenting the file entities (functions, typedefs, global variables…etc). In the other hand, all functions must have technical explanations documented using inline comments within the function body.

The following are examples of documented methods using Doxygen style in the lcd.c file that you used in your labs:

```
/**
*     This method will be used to print a single character to the lcd.
*     @author Dr.Zhao Zhang
*     @param chrData    The character to print
*     @date 2/28/2009
*/
void lcdWriteAChar(char chrData)
{
      PORTA |= ( data>>4 );
      lcdToggleClear(1);

      PORTA |= (data & 0x0F);
      lcdToggleClear(1);
}

/**
*     This method will be used to print a string to the lcd.
*     @author Jafar Al-Kofahi
*     @param chrString  The string to print on the LCD
*     @date 2/28/2009
*/
void lcdPutAString (char * chrString)
{
      ///This will hold the passed string length
      int intSize = strlen( chrString );
      ///This will be used to index the passed string
      int intIndex;

      ///Printout all characters of the passed string
      for ( intIndex = 0 ; intIndex < intSize ; intIndex++ )
      {
            PORTA |= ( data>>4 );
            lcdToggleClear(1);

            PORTA |= (data & 0x0F);
            lcdToggleClear(1);
      }
}
```

Notice how and what we described in the function headers documentation (recall the Documentation section of the coding standard guidelines document), we described the functionality that the functions will provide without mentioning any technical details (remember guideline #1 for function headers), then within the *lcdPutAString* function we described each variable we used in the function and what it is used for, and we explained what the for loop will do.

Sometimes, someone (same developer or another one) might have an improvement (performance, more readable, lesser code, reusing existing code…etc) over the existing code. In such cases, the developer must add the new code after commenting out the old code, the old code MUST not be removed but instead it should be commented out till we make sure that the new code perform the same desired functionality of the previous code, and the developer should document why he/she did the change by describing the reasoning behind the change and what problem it would solve or what it

would improve. But before describing the change the developer should put the date of the change as a time line reference and his/her name, so that if someone wanted to discuss the change he can know with whom to talk too. The following code segment is an example of such scenario (notice the for loop for printing out the characters).

```
/**
*     This method will be used to print a string to the lcd.
*     @author Jafar Al-Kofahi
*     @param chrString  The string to print on the LCD
*     @date 2/28/2009
*/
void lcdPutAString (char * chrString)
{
      ///This will hold the passed string length
      int intSize = strlen( chrString );
      ///This will be used to index the passed string
      int intIndex;

      /** 4/2/2009 Dr.Zhao Zhang: Instead of repeating code, and
      * since Strings are sequences of characters then we can use
      * the lcdPutAChar function */
      ///Printout all characters of the passed string
      for ( intIndex = 0 ; intIndex < intSize ; intIndex++ )
      {
            /** 4/2/2009 Dr.Zhao Zhang: used lcdWriteAChar
            * instead of repeating the same code to printout
            * the string value.*/
            /**
            PORTA |= ( chrString[intIndex] >> 4 );
            lcdToggleClear(1);
            PORTA |= ( chrString[intIndex] & 0x0F );
            lcdToggleClear(1);
            */
            lcdWriteAChar( chrString[intIndex] );

      }
}
```

Notice that the old code was commented out using comment block and the developer comments was in a separate block.

The following will guide you through out the process of creating the documentation document of your project. To run the program go to **Start -> All Programs -> doxygen -> Doxywizard**

The Doxygen main screen will come up (Figure 1)

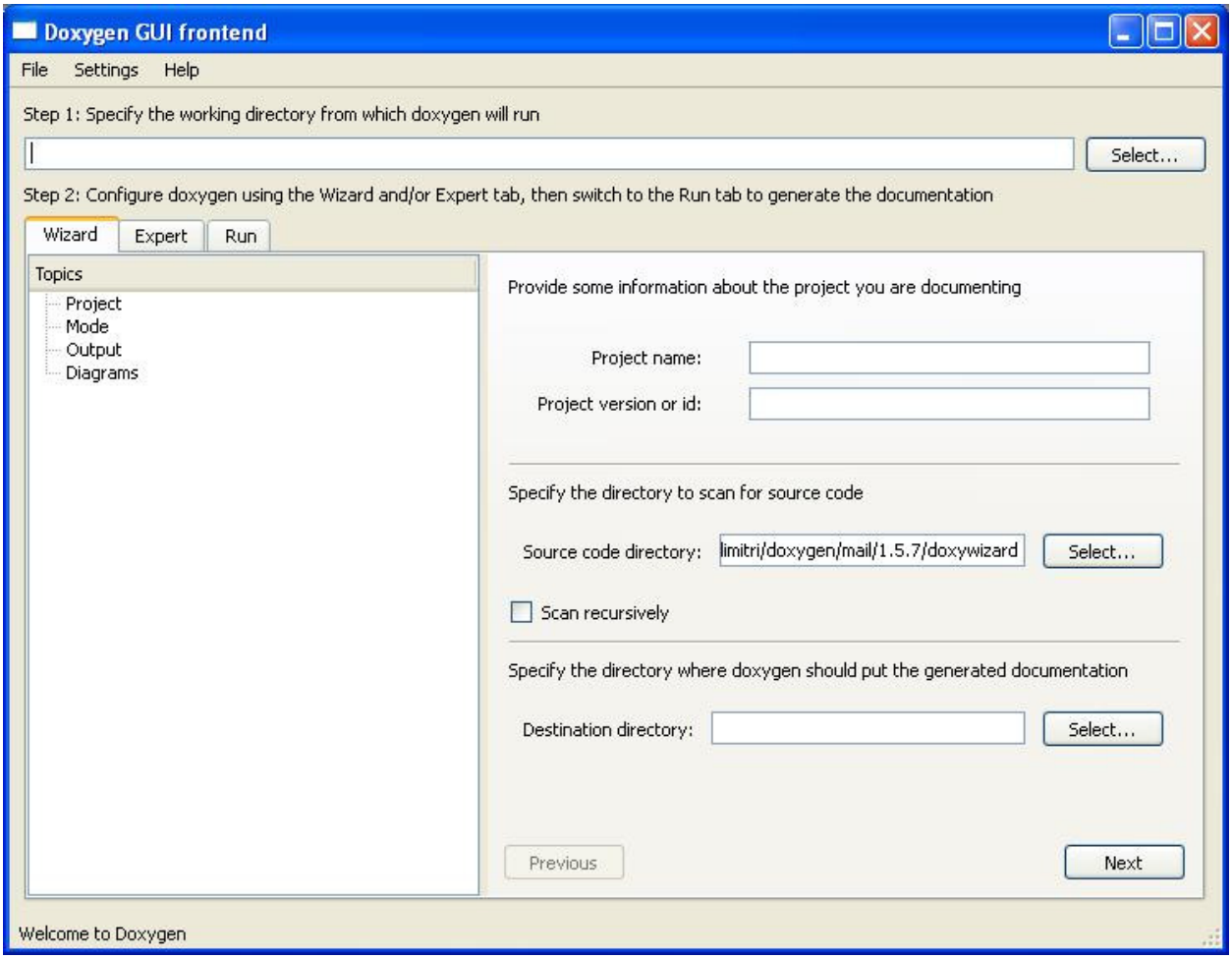

Figure 1. Main screen

You can use the tool by going through the Wizard steps or for more advanced options the Expert mode is available, for the requirements of this course you are only required and you must only use the **Wizard** when doing your project documentation.

In the **Project** screen in **Step1** you must browse to your Doxygen folder (e.g. C:\Program Files\doxygen), then for **Step 2,** you are required to give your project name and for which version the produced documentation will be, the source directory to read source files from, and the destination directory to which you want to output your documentation. Make sure you check the “Scan recursively” option to

scan all of your sub-folders. Your screen should be something similar to figure 2. After you finished entering all the necessary information press **Next**.

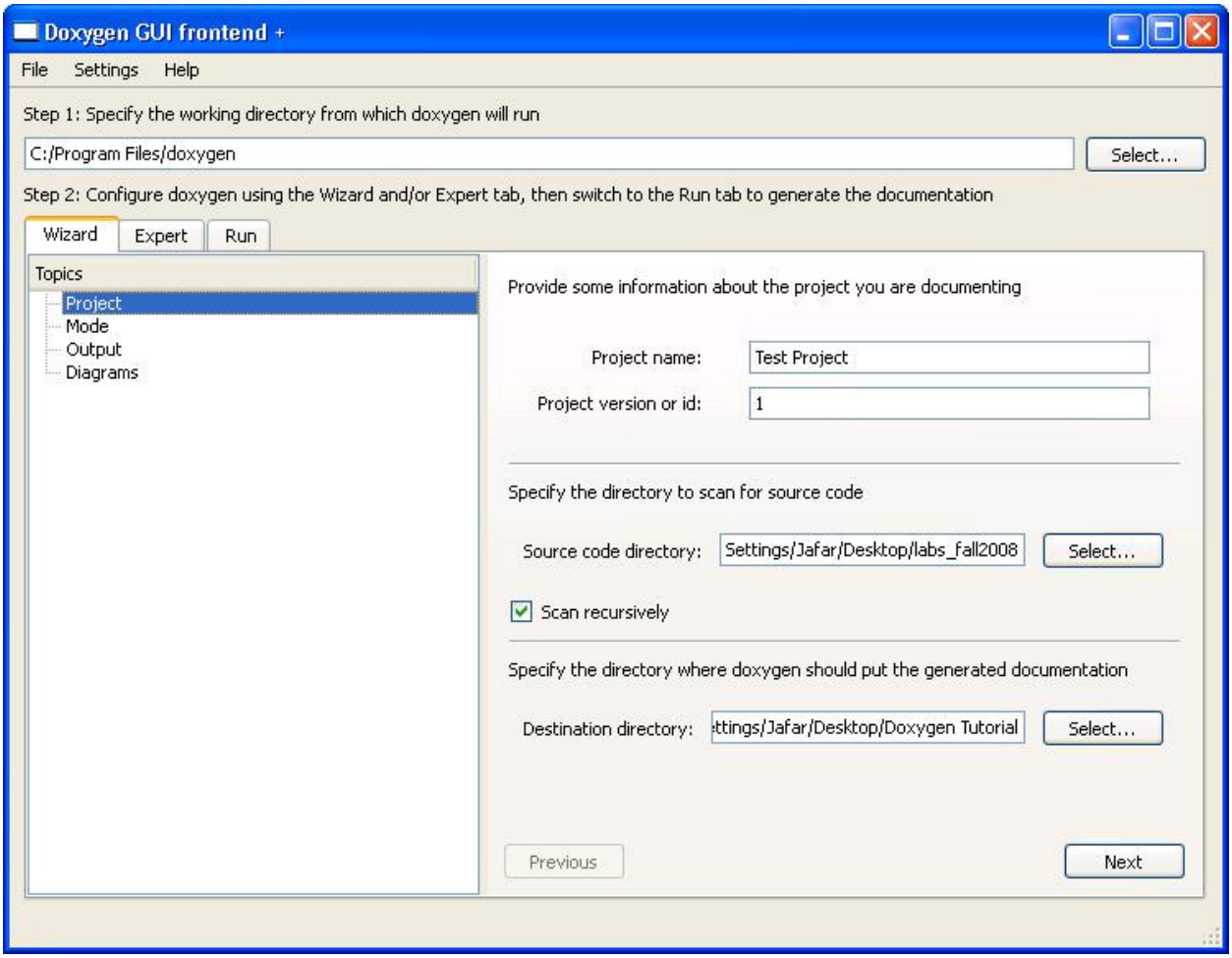

Figure 2. Screenshot

The next screen will be the **Mode** screen (figure 3), from the "Select desired extraction mode" select "Documented entities only" and check "include cross-referenced source code in the output" to do cross-reference between source code and documentation. Then to optimize the wizard for C, select "Optimize for C or PHP output" from the "Select programming language to optimize the results for". Then press **Next**.

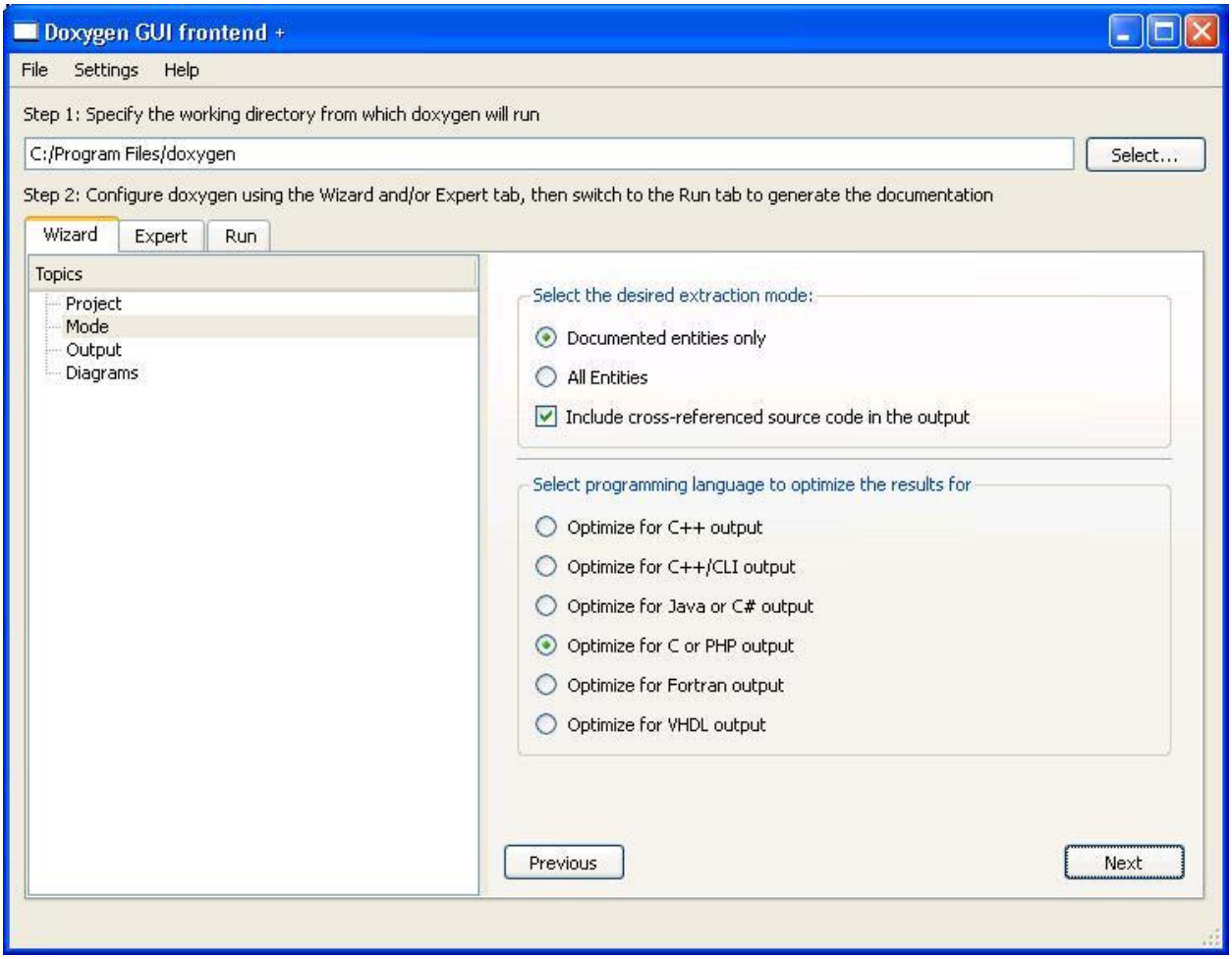

Figure 3 Mode Screen

After that you will be taken to the **Output** screen (figure 4), to choose the output type for your documentation. You will have the options to output your documentation to **HTML, LaTeX, Man, RTF,** and **XML**, for this course you must present your documentation as HTML with a navigation tree. To do so, make sure you check the HTML output format option, and then select “with frames and a navigation tree”

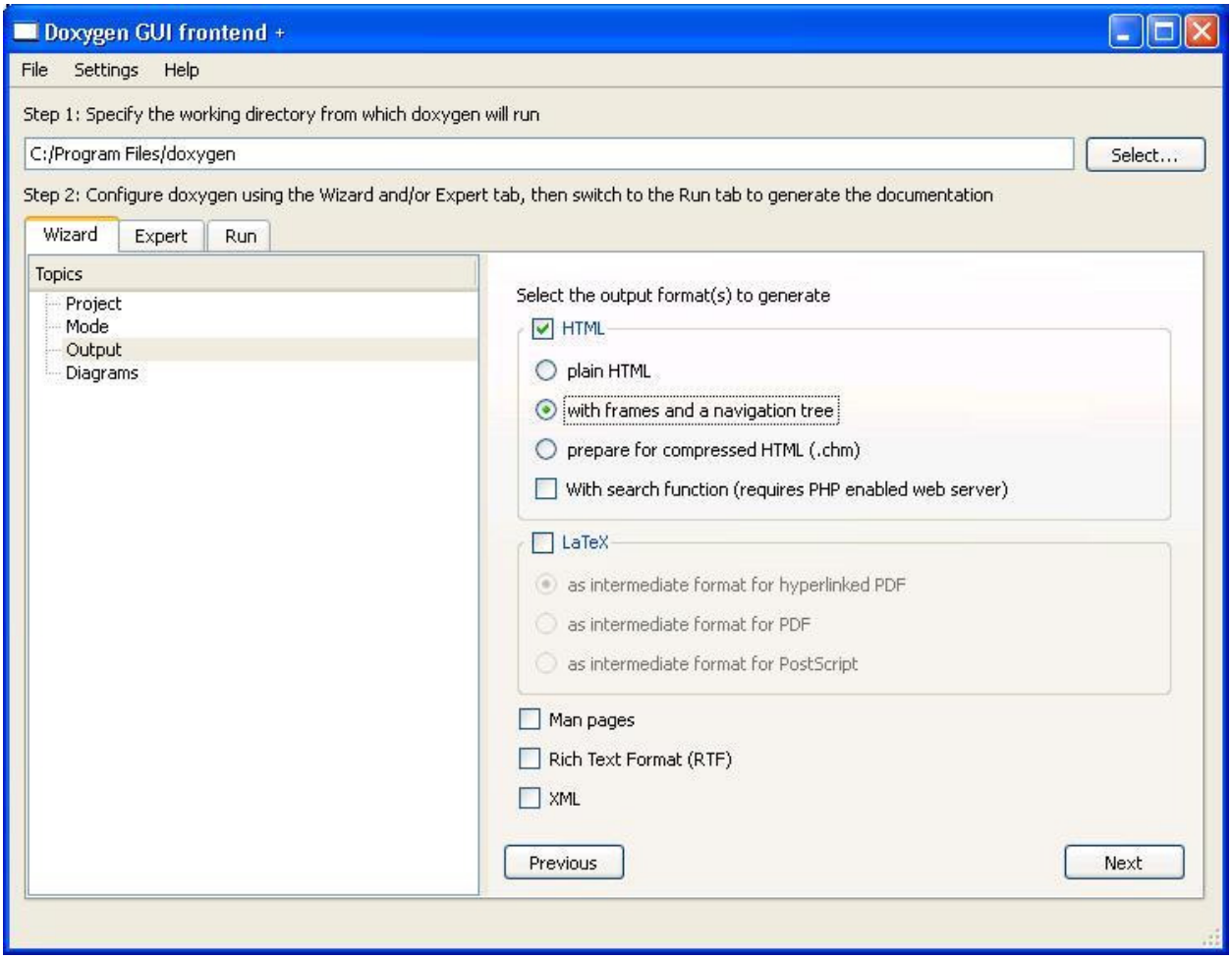

Figure 4. Output screen

With that done, the only thing left is to run the tool to generate your documentation, for this you need to click on the **Run** tab (figure 5), after going to the tab press the “Run doxygen” button and your documentation will be generated according to your settings. With this step done you would have produced the required documentation files.

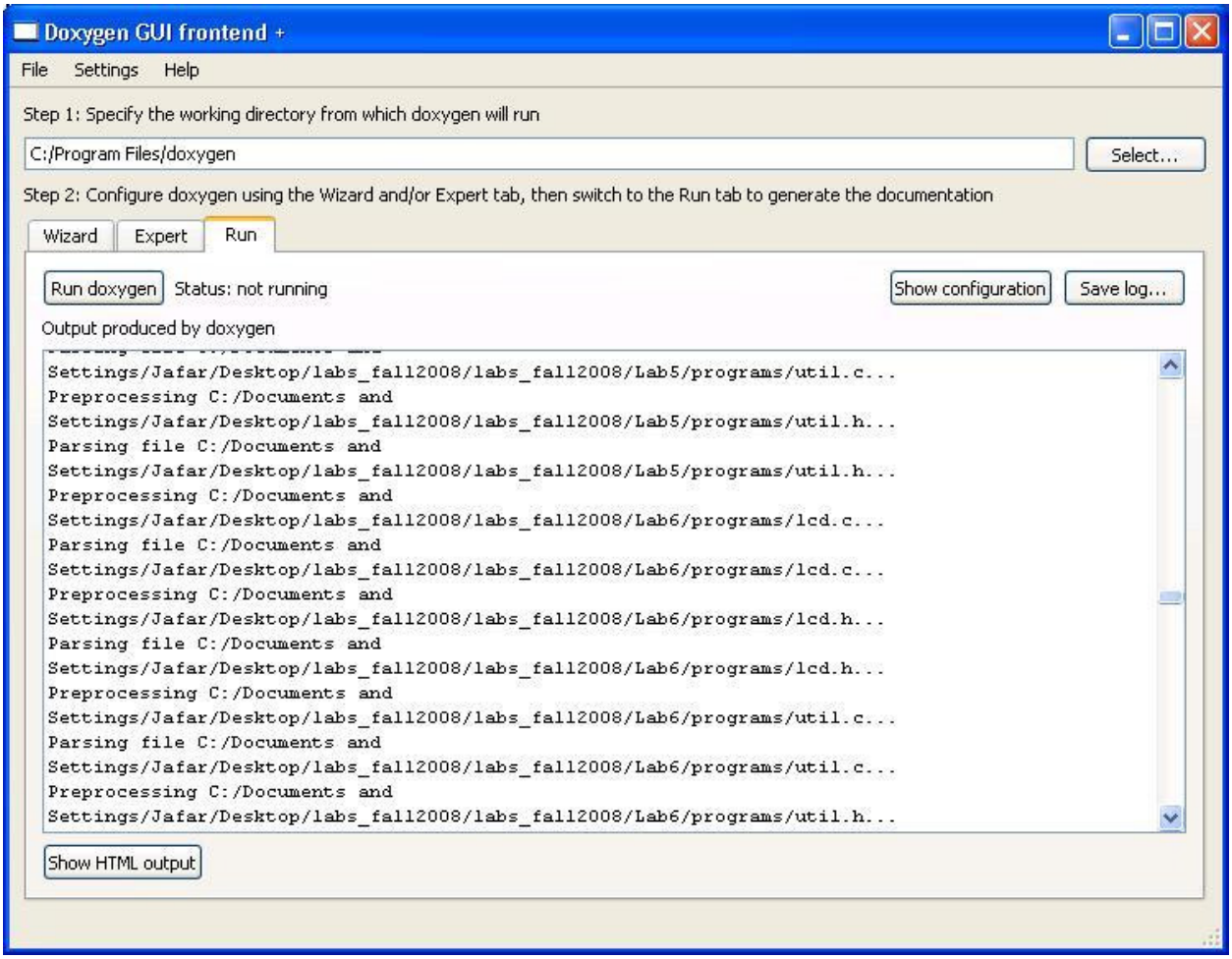

Figure 5. Run screen