

**Task**: create an application in which you build a list of tasks and execute them in multuple threads in a non-blocking fashion. See the picture above and description below.

# Description:

* Application should allow defining **functions** of one variable (double) that produce one variable (double) **in the code**. So one should be able to define a class that will define a function and it should be somehow registered within the application to appear in the list above (eg. by using MEF or other DI framework). You choose a way of registration yourself. It’s **not** required to be able to add new functions at runtime, you are allowed to recompile the application to add a new function.
* Each function should have a **display name** (string) and a **category** (string) (and of course a body which will actually compute the function)
* Function’s **display name** is used to represent it in the Functions listbox (B)
* Function’s **category** is used in Function category combobox (A). When you select a category, only the functions of this category should appear in the list (B)
* By default you should have the following functions with the corresponding categories: Sin (Trigs), Cos (Trigs), Tan (Trigs), Ln (General), Sqrt (General), Rnd (Special). The latter function should generate random number taking the function parameter value, casting it to int and using as a seed for random generator.
* Once you’ve selected a function in Function listbox (B) you should be able press button (C) to add a new task with this function. A new line corresponding to this task should appear in the list (E).
* Each task is a function associated with it, function parameter (double) and iterations number (int). When you select different tasks you should see their parameters
* Function parameter is set by the user in the textbox (F). Default value for it is 1,0 (one).
* Iterations number is the number of times the task will execute its function with provided paramter. Default value is 1.000.000 (one million).
* Once you have a task selected in the listbox (E) you can press button (D) to remove the task from the list.
* After you formed a task list you can press the button (I) to run the tasks for execution. All controls should be **disabled** except for progressbar and the button itself.
* Internally each task must be is executed **in its separate thread**. Again, the task execution represents calculating the function specified number of iterations. Function result is **not** used anywhere. Number of threads running at any time must be limited to machine processors count (Environment.ProcessorsCount). So on 4-core processors your tasks should be executed in 4 threads at any time in a round robin fashion (or 8 if HyperThreading is there).
* The progress of the execution should be reflected in progressbar (H). It should show percent of work done (you decide how granular this progress should be), and the number of tasks executed of total number of tasks.
* After pressing the Run button (I) it should turn into a Stop button which goal is to abort calculationat any time. So execution should be non-blocking. After pressing Stop button execution should stop and all controls must be re-enabled. The progress bar should not be reset showing the last value before abort.

# Notes:

* **Main goal is to finish the task in time even if you had to simplify some things** in order to have it done in time. For example if you feel too uncomfortable with limitation of multiple threads, you can use one thread per task so for 100 tasks you will have 100 threads. Or if you feel too uncomfortable with multithreading itself you can execute all tasks sequentially but at least make application not block the UI.
* In terms of implementation of functions it should be a convinient API for implementing new functions and registering them.
* In terms of UI try to use Bindings and DataTemplates as much as possible to quickly achieve good results
* If you have time make UI checks preventing the user from providing error input and doint bad actions.
* If you have time provide a way to indicate completion of individual tasks so the user can see that certain tasks have finished (for example by coloring its background in green)
* The following things would be appreciated (but not required):
  + Clean well-formatted code with logical classes, interfaces and method names
  + Using TPL and/or Rx for multithreading and asynchrony
  + Using at least one Dictionary
  + Using Bindings (WPF)
  + Using DataTemplates (WPF)
  + Using interfaces, classes and inheritance
  + Checking input parameters
  + Using concurrent collections (ConcurrentQueue, BlockingCollection, ConcurrentBag and so on)
  + Using synchronization primitives
  + Handling exceptions when it’s needed and gracefully handling them without crashing the application

Good luck!