# *Welcome to ProfileSharp. The next-gen code, performance and memory profiler for your .NET applications.*

## Here is the list of contents :

- Initializing the profiler.
- Starting profiling. When and how to.
- Stopping. When and how to.
- Saving results. Clearing the profiler's cache.
- Closing down.

- **Memory Profiling [*Steps to follow.*]**
  - Starting
    - Configuring the profiler. Choosing what you exactly want the profiler to profile.
      - Applying filters.
      - Runtime options.
      - Garbage Collection
  - Selecting the application to profile.
  - Initializing the profiler.
  - Capturing state of the .NET heap. When and how to
  - Saving results.
  - Clearing the cache. Closing down, and some things to remember.

- Analyzing
  - **Performance Analysis [Steps to follow]**
    - Opening a saved session.

- Seeing the profiler configuration settings that you had chosen for the session.
  - Choosing and applying an analysis view [report]
    - View % Time Consumption and Hit-Count
    - Analyze Performance by Threads and Modules
    - Analyze Exceptions and exception stack trace.
    - Analyze functions by parent functions
    - Analyze functions by child functions
    - Analyze performance at source-code line level
- Memory Analysis [Steps to follow]
  - Opening a saved session.
  - Seeing the profiler configuration settings that you had chosen for the session.
    - Choosing and applying an analysis view [report]
      - Analyze In-Memory Objects (individually)
      - Analyze In-Memory Objects (grouped by Object-Class)
      - Analyze In-Memory Referenced Objects
      - Analyze In-Memory Referencing Objects
      - Analyze Objects by allocation
- Comparing Memory Sessions

## Performance and Memory profiling and analysis. A description.

How often have you found a piece of code or a program module behaving abnormally or leaking memory when you have applied all the knowledge you have, to make it work properly. Or for that matter how much time on an average do you have to spend to fix a memory leak or to improve the time consumed by your program to make it run faster. Or how do you gain the know-abouts of the internal working of your application, be it memory related or the response and execution time. The answer to all these trivial and often-faced yet indispensable questions is ' Profile your application '. Or synonymously, Use **ProfileSharp**.

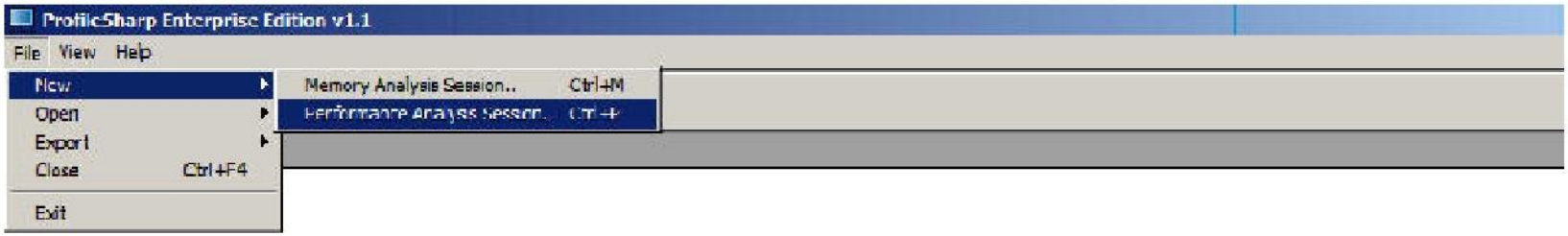
So what actually is profiling, and how can you benefit from it.

Profiling is inspecting your program for memory leaks and performance bottlenecks, and figuring out or pinpointing the hotspots in your program , and where they actually lie. While memory leaks or performance throttles can be identified using many of the traditional ways , but it is quite often that programmers resort to the hit and trial method ; ...make slight changes to the code and test for leaks or performance using many of the tools available in the market that produce runtime statistics for your program, one that is very popular is the '**Perfmon**' that ships with the Windows OS. While this strategy at times does lead to results but can be very exhausting and inaccurate. Here steps in '**ProfileSharp**', the most robust ,accurate and result-oriented .NET Memory , Performance and Code Profiler. With '**ProfileSharp**' besides you, it will only be a matter of few minutes to hunt down the memory and performance hotspots in your code or for that matter any .NET component you use, regardless of the fact , whether you have the source-code for it or not, or your application is a debug* or a release build, which otherwise could have taken hours or even days to figure out and fix. What's more, you will never need to recompile your program at any stage. You can directly get on with profiling your .NET application with '**ProfileSharp**'. So lets take a step-by-step tour studying the valuable features that this piece of meta-software called 'ProfileSharp' has got, and how to apply them to your needs.

Performance profiling. Steps to follow.

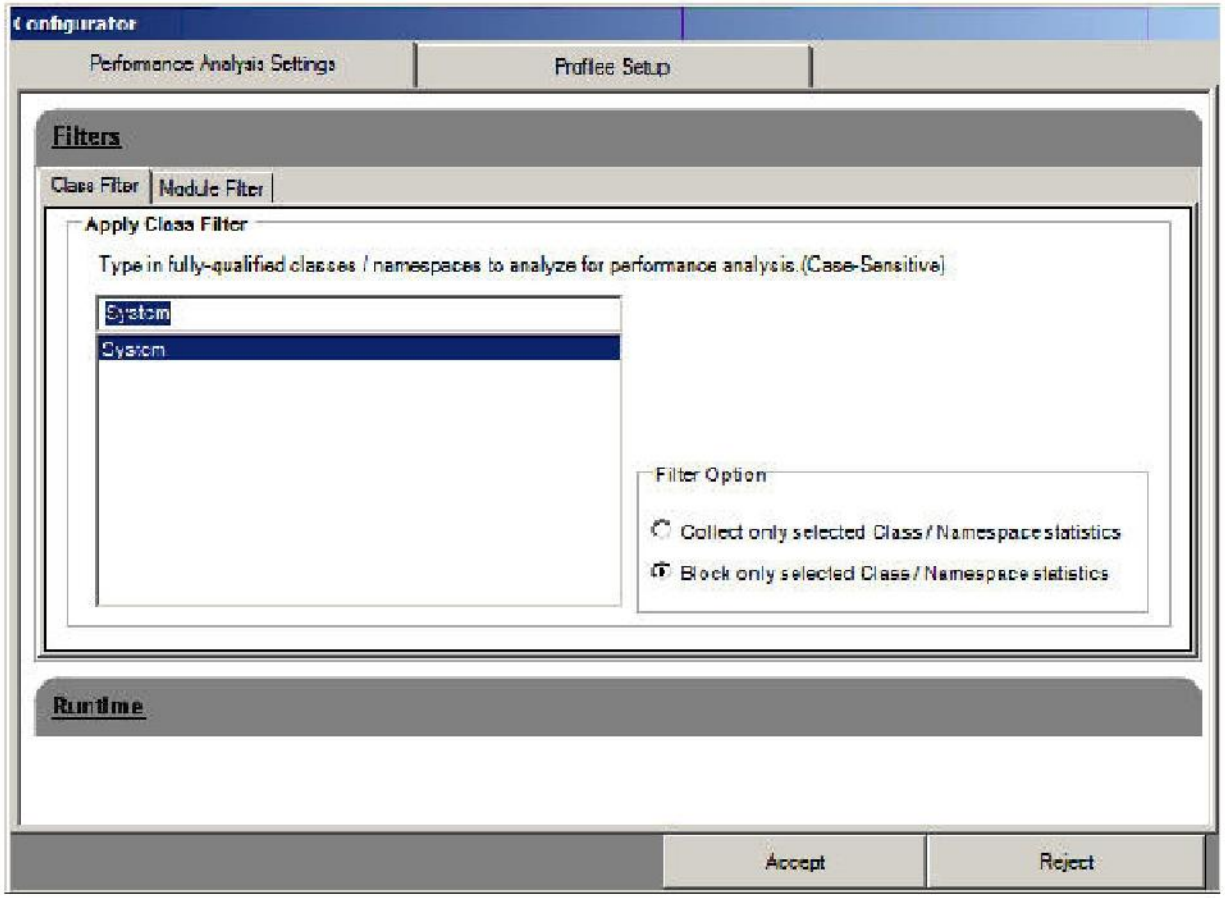
**Starting.**

To start the profiler for performance profiling, select the menu '**File->New->Performance Analysis Session..**' or press Ctrl+P. This will open a dialog box where in you can configure all the necessary settings and configurations for the data that you want the profiler to collect when it profiles the target application. This dialog is called the '**Configurator**'.




Click the '**Filters**' tab. The tab will slide open. The filters tab is use to apply Data Collection filters to the data that will be collected during profiling. You can apply both Namespace and Module filters. Type in a namespace [*Case-Sensitive*] in the combo box and press enter. This will add the namespace to the list ['System' namespace by default]. All the functions that belong to any of the selected namespaces, will now not be profiled. Or you can click the radio button, labeled 'Collect only selected class/namespace..' at the bottom of the tab, if you want only the listed namespaces' or classes' functions to be profiled. You can enter Namespaces as well as Classes [Fully-qualified, case sensitive] e.g. System.Data. DataSet (class) or System.Data (namespace). To remove an entry, select a namespace/class and press 'Delete'.

Similarly you can enter the module names in the 'Module filter' tab, that you want/do not want to be profiled. Module names are case-insensitive. Only module names are required , not their full paths. So .e.g. you want to profile MyNameSpace,MyNameSpace2.MyClass and MyDll.dll, enter My Namespace and MyNameSpace2. MyClass in the Namespace/Class filter and select the option "Collect Only selected class/namespace..." and in the Module filter, type in mydll.Dll and select the option "Only profile selected modules..". Or say you want to exclude System and Microsoft namespaces from profiling , type in the namespaces and select "Block selected class/namespace..." . In case of a collision , Module filters take priority over Namespace/Class filters




Runtime Configurations for Performance Profiling

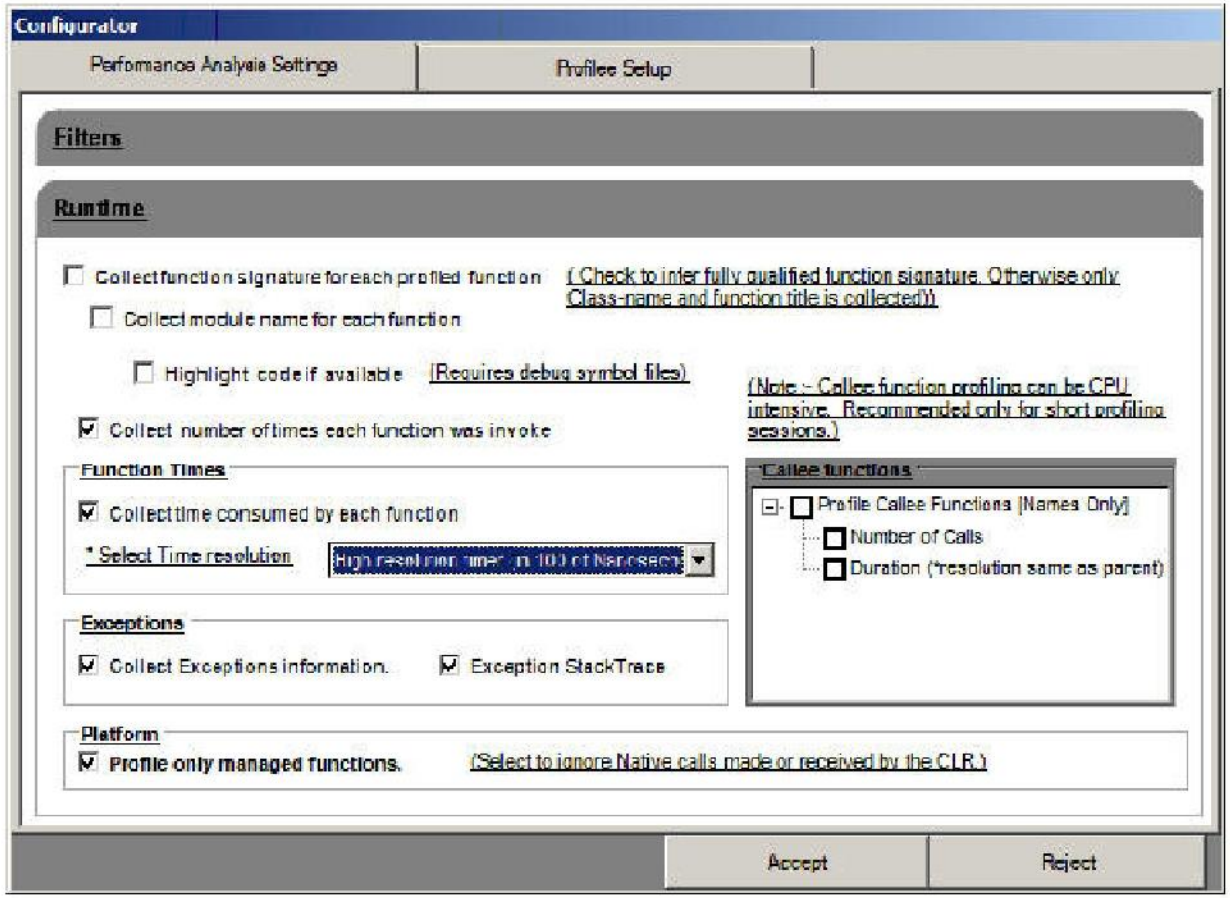
Click on the '**Runtime**' sliding-tab to select various types of information that the profiler should collect during performance profiling.

While most of the options available are self-explanatory, like , whether you want the profiler to collect module names of all the executing functions in your application and group the functions accordingly during analysis. Similarly, selecting 'Profile only managed functions' option profiles only purely managed to managed (CLR<->CLR) function calls, otherwise native [COM-Interop and P-Invoke] function calls are also profiled. [Please note that only native function calls directly made or received by the CLR (of course, along with the normal CLR to CLR calls) are profiled in this case, i.e. CLR<-> CLR , CLR->native or native ->CLR function calls are profiled and not native<->native ]

To profile your application right to the source-code line level, select ' **Highlight code if available'** option. Note that this option is mutually exclusive to **'Exception Tracing'.** i.e. you can not profile to line-level with **Exception Tracing** selected and vice-versa.

To let profiler collect the information about callee/caller functions [child and parent functions to a function] select the corresponding options in the **Callee Functions** window, to the right of the **'Configurator'.** This option collects information about the function call hierarchy, i.e. which function *called* / *was called by* which of the other functions and in which order.

The default configuration is what you would ideally want to profile in most cases, unless you have specific requirements.




Selecting the application to profile

Click on the '**Profilee setup**' tab of the ***Configurator***.

**Profiling .NET Windows Services, .NET Enterprise/COM+ surrogates (the dllhosts), ASP.NET and other background processes :**

Once you have selected the profilee tab, you will see 4 options, each pertaining to a specific set of application types you can profile. For background

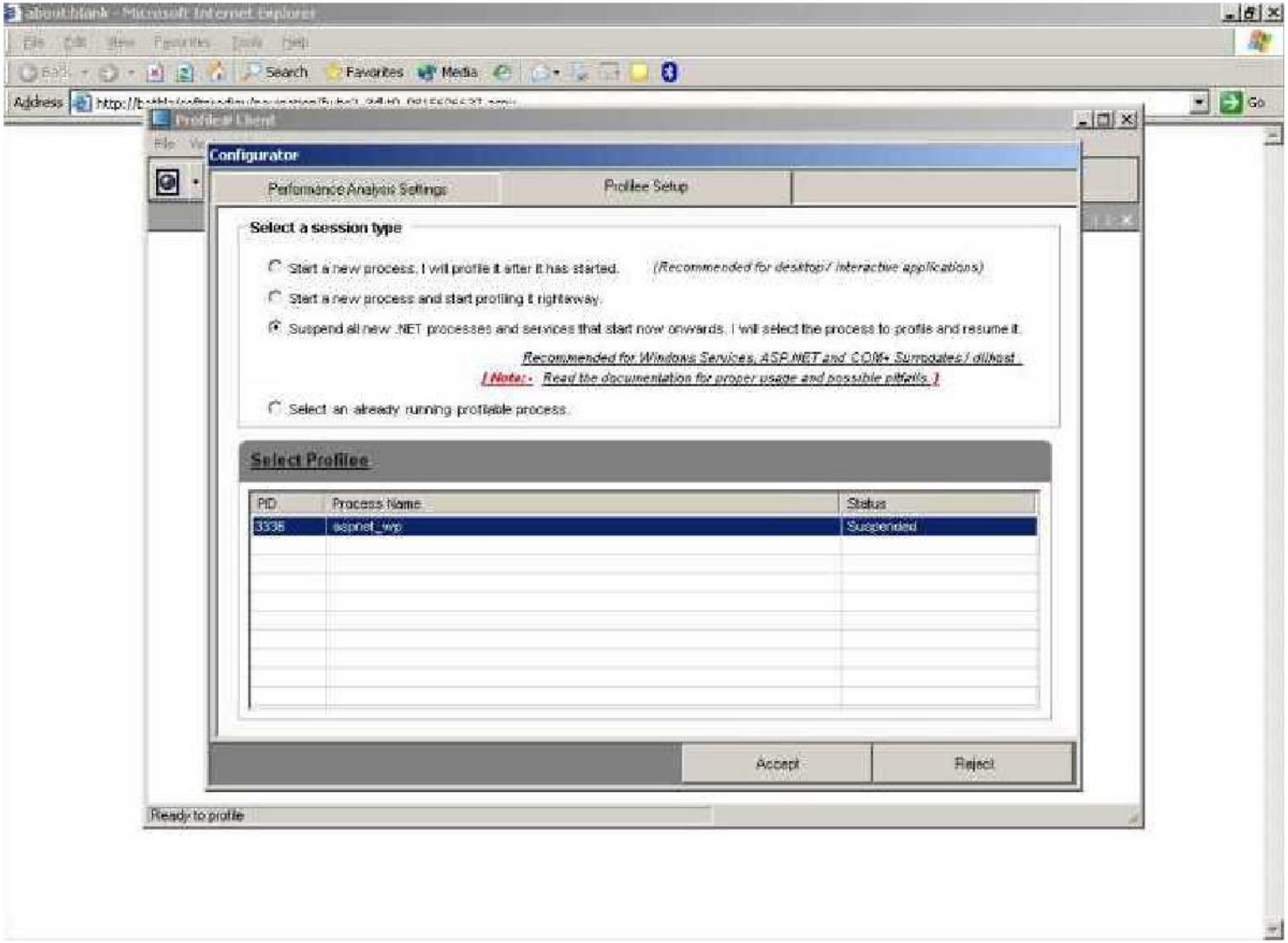
processes and services as dllhosts, ASP.NET etc.the bottom 2 options (option 3 and 4) are viable. By default, option **3** i.e. **'Select an already running profilable process**' is selected. You can also select the option **4** i.e.**'Suspend all new .NET processes and services....**'.if you want to profile your .NET service/surrogate/ process or ASP.NET to be profiled right from the first .NET method called (i.e. from scratch).The later option when selected will pop-up a dialog to confirm you want to profile the background process right from when the CLR is loaded into the process. Once this option is selected , the profiler will suspend any new .NET process thus started , so that you can initialize the profiler even before the CLR is. The profiler will attach to any new .NET application or process, while the **Configurator** dialog box is open and one of the **last** 2 profilee selection options is in selected state.Please make sure that you do not close the **Configurator** or unselect the .NET Service profiling option (one of the option 3 or option 4) at this time as the profiler is monitoring any new .NET process that starts so that it can attach to it rightaway. Now press **'Win +D'** to minimize the profiler along with the **Configurator** dialog box and **stop** the process/service or reset the IIS to stop ASP. NET for once (if you are profilng ASP.NET). Now launch the client application that re-spawns your .NET windows service /dllhost or the web page that initializes your ASP. NET process. Once the target profilee process is started you can restore the Profiler window to see the status of the service/process/ASP.NET either as running or suspended (depending upon you selected option 3 or option 4 , respectively ).

Please note that the profiler can only attach to those .NET processes/services/ASP.NET that have been stopped and restarted (explicitly or implicitly by a client application) while the **Configurator** dialog box is still open and either option 3 or option 4 is in selected state.(This is called the profiler's **Monitoring state**). Once you close the **Configurator** or un-select option 3 or 4, the **Monitoring** state is terminated.

Once you see the profilee process/service name listed in the list box contained in the **'Select Profilee**' sliding tab along with its current status (**Suspended** or **Running**) you can select the process and click the **'Accept'** button to start the profiler.

Another point that must not go unnoticed is that when you profile a .NET windows service with the option '**Suspend all new .NET processes and services..' ,** you must select it, attach to it and resume it within **30** seconds of after it was spawned (in the suspended state) as the SCM (Windows Service Control Manager) stops the service if the service is unable to attain **'Running'** Status within that time. (See FAQ portion of

the documentation for more information on this topic) **To reiterate:-**The profiler can profile only those .NET application/process/service etc. which were started while the profiler was **running** and in particular it can profile a **.**NET windows service/dllhosts/ASP.NET process only if the process was stopped & re-started (implicitly or explicitly) while




the Configurator dialog box was open and either option 3 or option 4 was in its selected state (i.e. the profiler's **monitoring** state)

**Profiling .NET Windows/Desktop applications :**

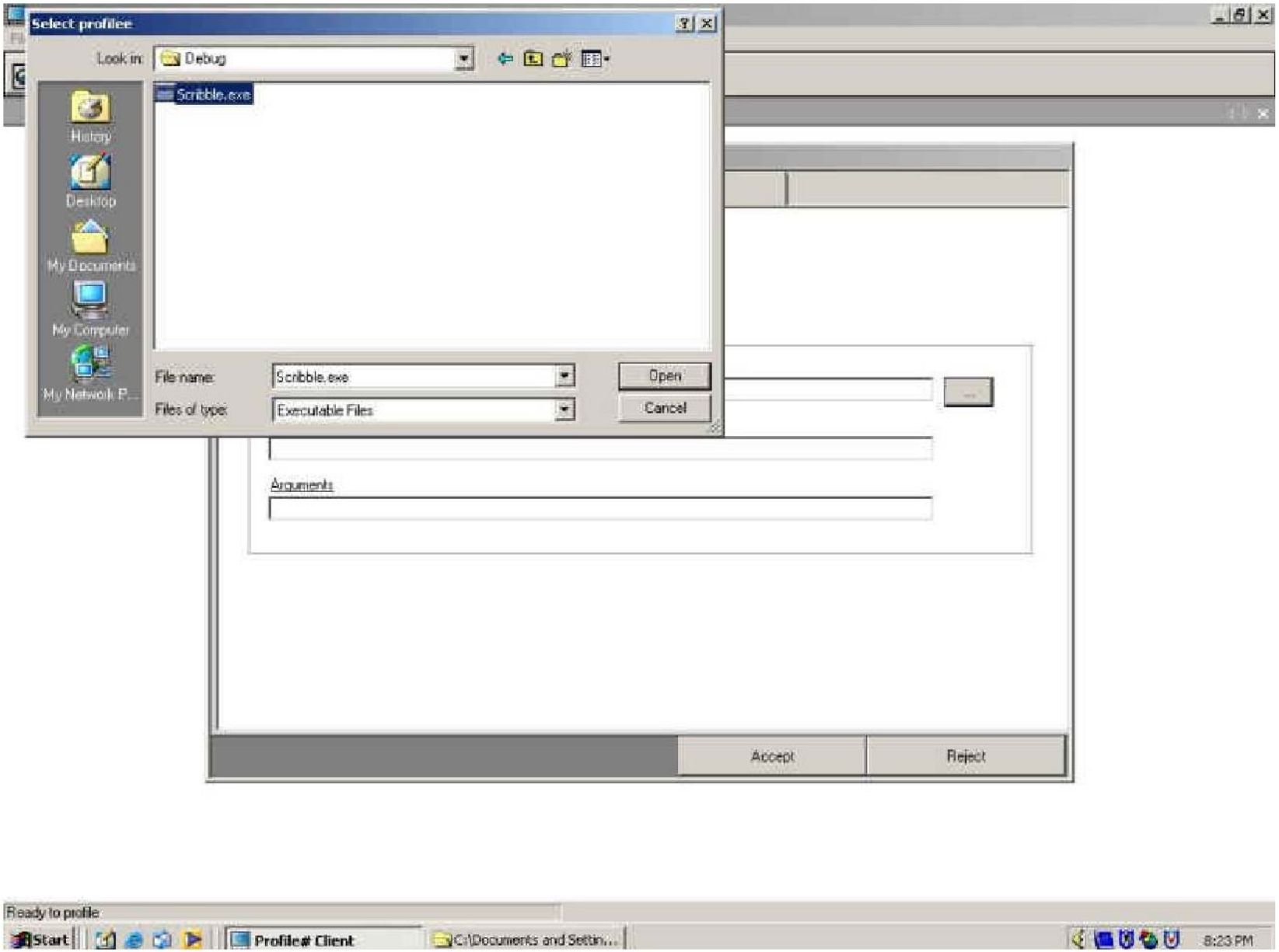
Following the same pattern as that used for profiling .NET Windows services/dllhosts/ASP.NET , we can profile .NET desktop /windows applications. To do so select either option **1** or option **2 (**instead of option **3** or **4).**

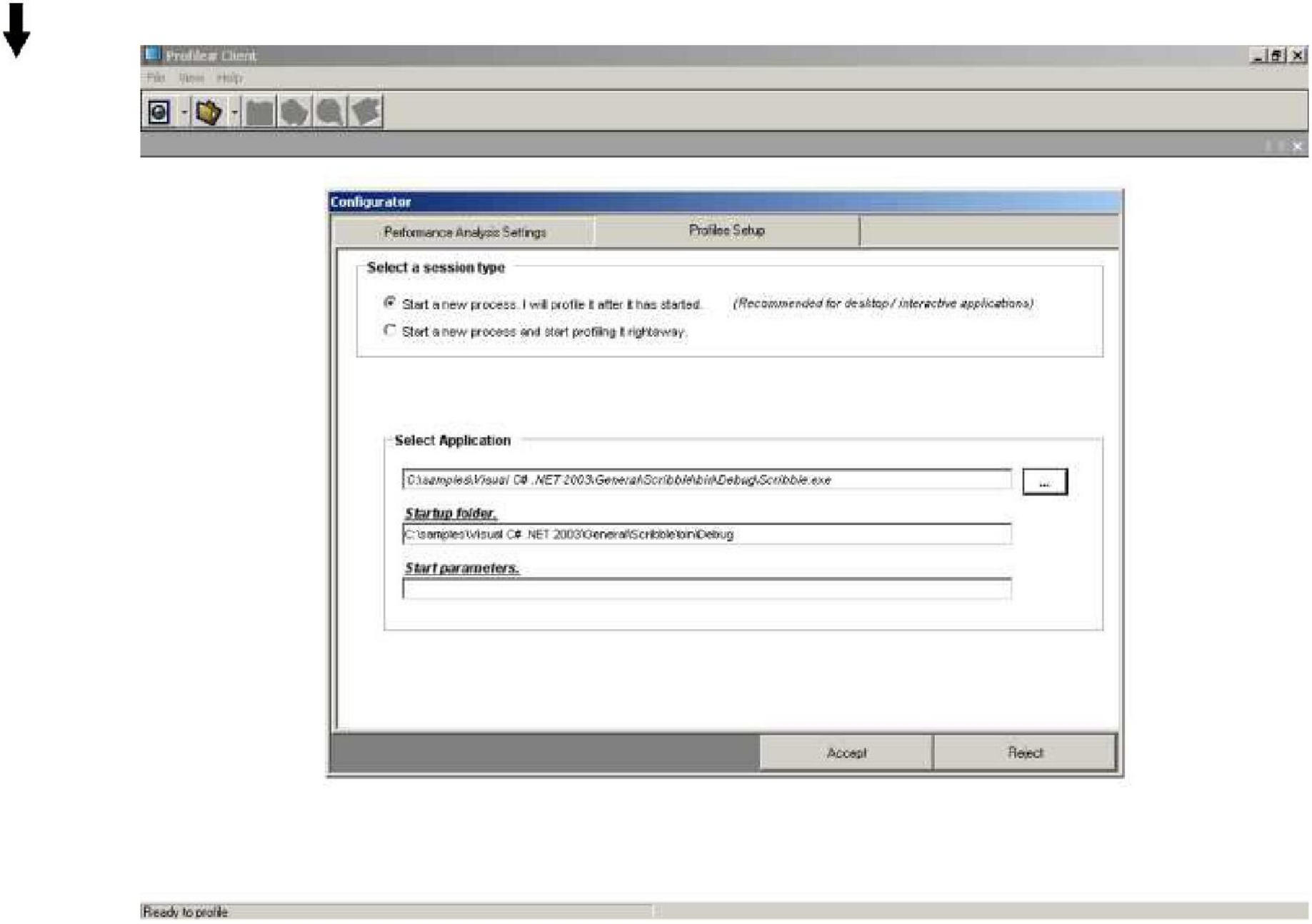
**Click on the '...' button to open the File-Open dialog. Browse to the application's executable that you want to profile and select it. The default** application directory for the executable will be automatically filled in. You can change it to meet your requirements]. At the '*Arguments*'

textbox you can specify any command-line parameters that your application requires. Or you can leave it empty if there are none.

If you want the selected application to be immediately profiled as it starts, select the option '**Start a new process and start profiling it right away**' **else otherwise. This option when selected will start the Profilee application in suspended mode so that you can start profiling it eve**n before the CLR has actually loaded into the process. This helps you analyze what goes inside your .NET application when it is starting.

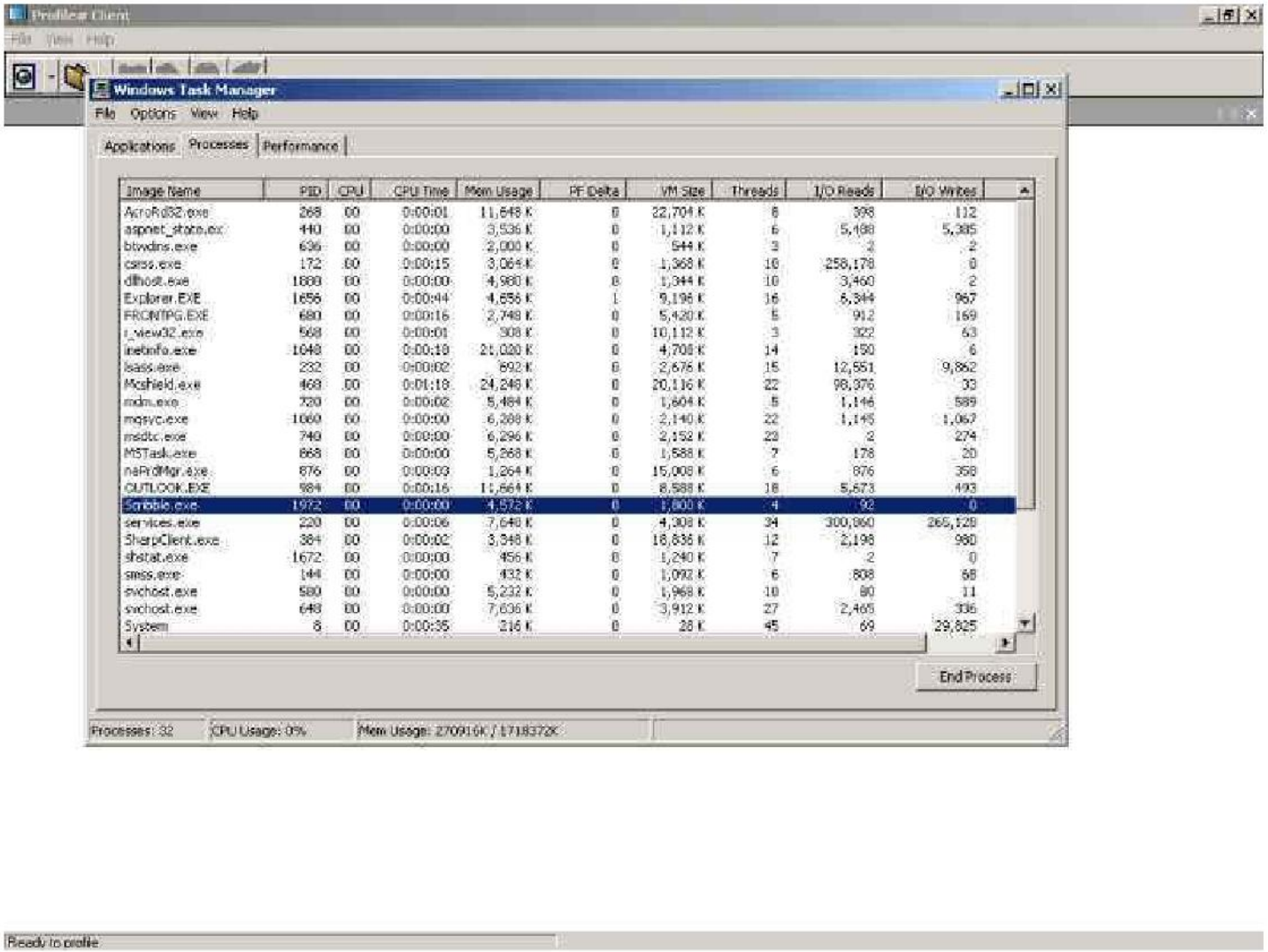
Click the '**Accept**' button to start the profiler.




Profiler Client
File View Help
Configurator
Performance Analysis Settings
Profiler Setup
Select a session type
Start a new process. I will profile it after it has started.
(Recommended for desktop / interactive applications)
Start a new process and start profiling it rightaway.
Select Application
C:\samples\Visual C# .NET 2003\General\Scribble\bin\Debug\Scribble.exe
...
Startup folder.
C:\samples\Visual C# .NET 2003\General\Scribbleton\Debug
Start parameters.
Accept
Reject
Ready to profile

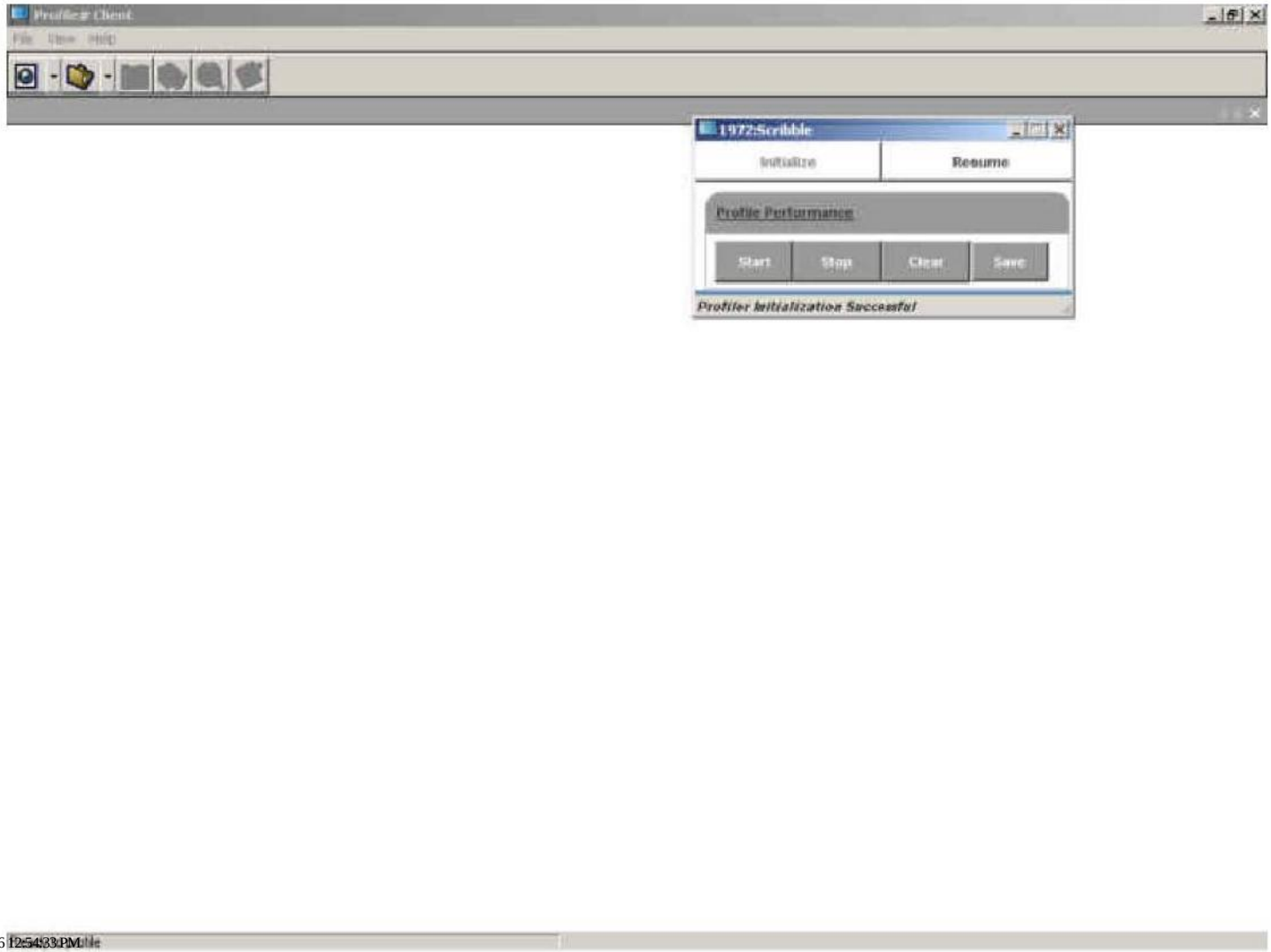
#### Initializing the profiler

Once you have clicked the '**Accept**' button in the Configurator, your application will be started as normal unless you had selected the profilee application to be profiled **immediately** (i.e. in suspended mode). In this case, your application has actually started but the profiler has suspended it so that you can profile it even when it is starting up. You can confirm it by running 'Task-Manager' and clicking the 'Processes' Tab to see your process in the list.




You will on your screen now see a small window , titled with a number and the name of the executable being profiled. The number is the ProcessID of your application. Click '**Initialize**' button on the window [This is always the first step you will do before profiling any application.] Clicking this

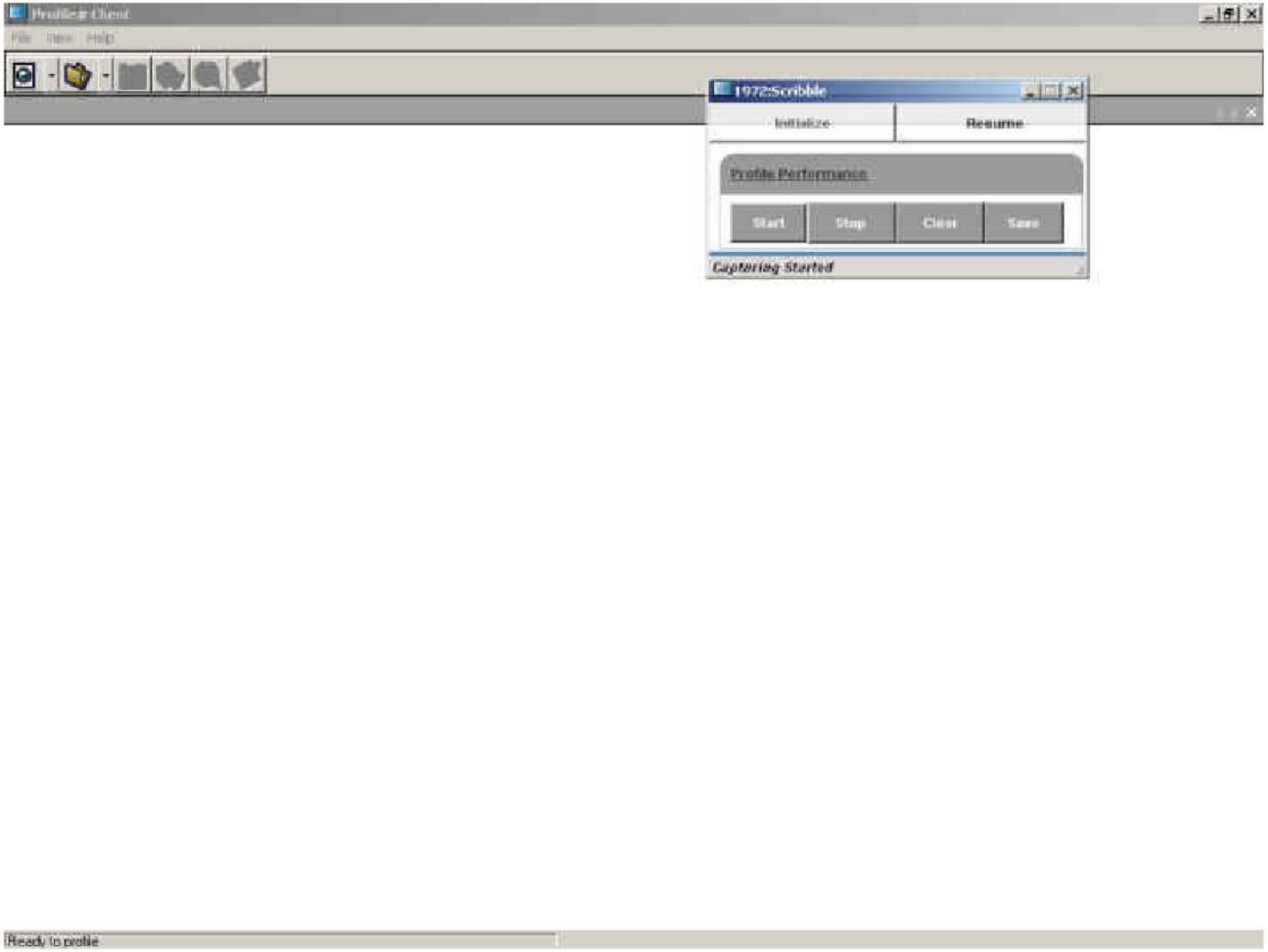
button ensures that everything is fine and the profiling environment has been successfully loaded into your application, other wise it will pop-up an error message describing what caused the failure.



**Starting to profile**

**No profiling has been started as yet**. Now that initialization is successful, we can get on with profiling. Click the **Start** button to start profiling. Please note that if you click the '**Resume**' button first, you application will be let-off by the profiler from the suspended state and you will not be able to profile the application's start-up flow. You can however always do that if you want it that way. So first click the '**Start**' button.

Once you press the **Start** button, the profiling is turned on. Now click the '**Resume**' button. Your application will begin resumption, with CLR being loaded into it while the profiler is actually collecting all the data it was configured to collect during application initialization . Depending upon your application, it may take a few seconds to few minutes before it completely resumes.



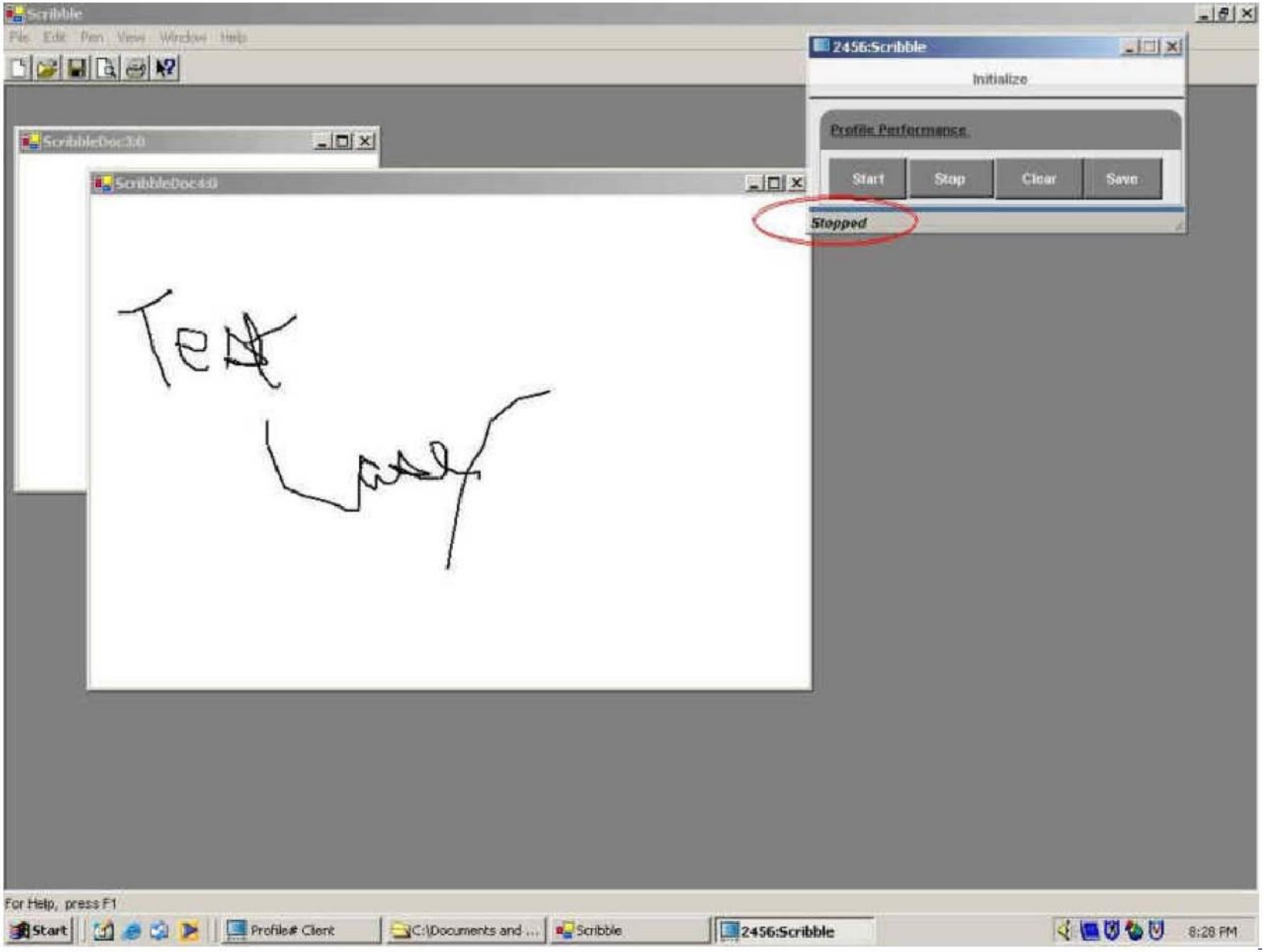
Now once you see your application up and running you can either decide to continue profiling along and do the necessary steps to execute your program-flow as you would normally do or you can instead optionally stop profiling by clicking the '**Stop'** button and rather save the results collected by the profiler up to now (by clicking the '**Save**' button).

In case you continue to profile, you may experience some slow-down in your application at this point. This is because the profiler is collecting profiling information while you are working. But do not worry. The profiler very accurately adjusts the information it is collecting , and takes into account the time spent by itself to profile the application. So the results it 'd save and produce for analysis later [as we shall see], are very very accurate and match the real-time statistics of your program.

---

**Stopping**

Once you have performed the test-case / program flow , click the **Stop** button to stop the profiling. At this time ,the profiler is holding-up all the profiling information for the particular flow you have just completed, in its memory.

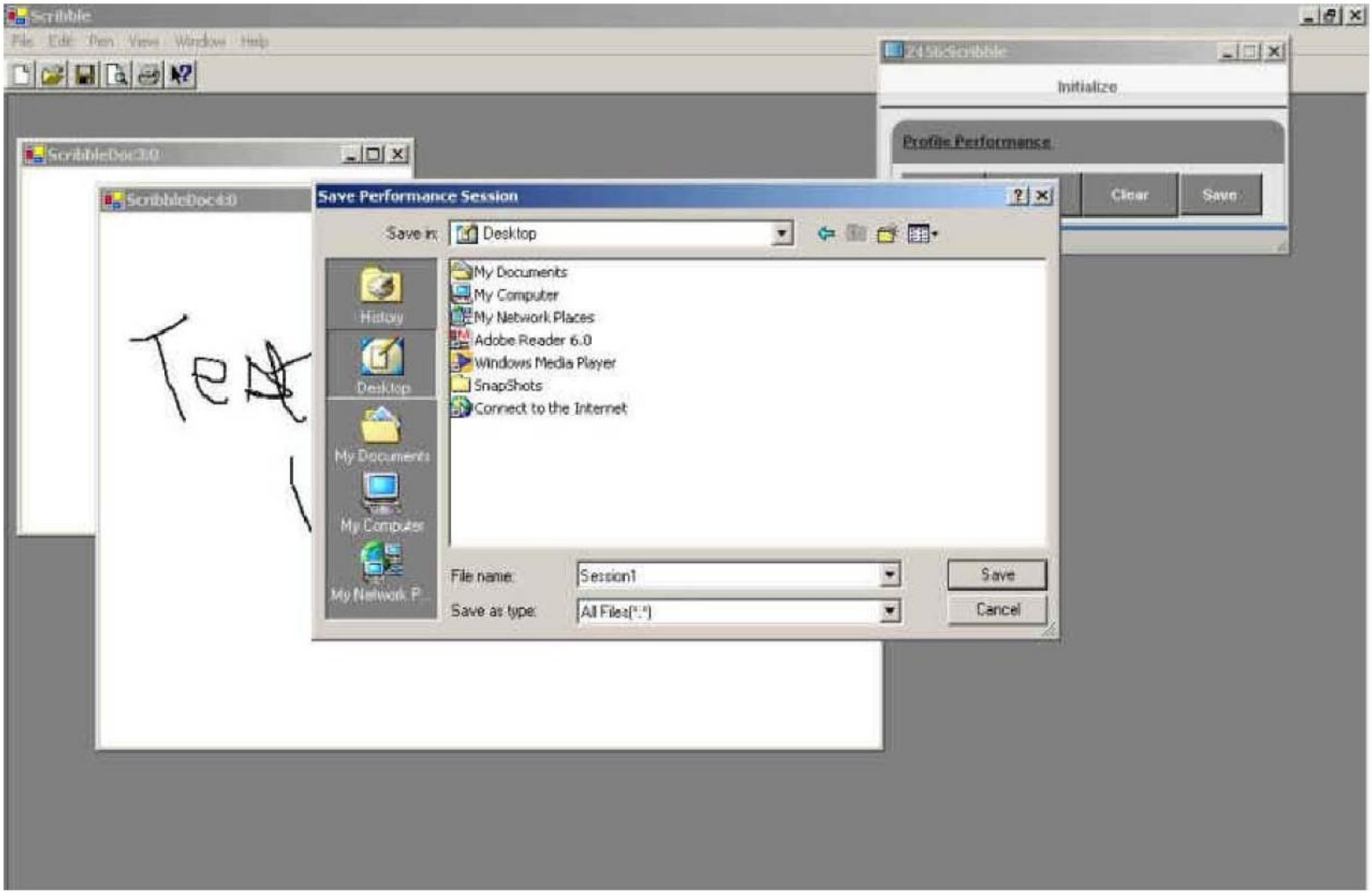



**Saving the collected profiled data**

Click the **Save** button to save this information [the *Session]* to your hard drive for later analysis. Please note that at this point if you click the **Start**

button [ or do any other action with the profiler for that matter] rather than the pressing **Save**, all the information collected up to this point by the profiler will be cleared from its memory, and the profiler will be ready to profile another session afresh. So save it first !. Also please note that it is highly advisable that you save all the sessions to one directory, or at least choose unique names for all the sessions that you profile and save. Otherwise the profiler may not be able to distinguish between 2 sessions later on when we analyze the sessions, and would give duplicate or skewed results.

Please note here that you must save the profiling data to a directory that has ***Write*** permissions enabled for the user account, the ***profilee*** application/ process (and not the profiler) is running under.This scenario is most relevant in case you are profiling ASP.NET which by default runs under the account of the user ASPNET. The best thing here would be to save all the profiling sessions to a directory that has write permissions enabled for ***Everyone***. Otherwise the session may not be saved giving an *'Access Denied'* error!. Ofcourse this issue is relevant only if you are saving the profiling sessons to a drive with **NTFS** file system. (See FAQ for more information)




### Clearing up for next session

Now that you have saved the session, you can press the **Clear** button to force the profiler to clear any persistent in-memory information it is holding. The profiler is now ready to profile another session.

You can now perform some other set of operations in your application to lead your application to the point where you want to start profiling it again. This can be a use-case or a particular flow of your program.

Please note that in case you had not selected to profile your application right from the start, you will not see a **Resume** button in the profiling window and your application will rather start-up immediately so that you can immediately '**Initialize**' and '**Start**' profiling it rather than **Resume** it first. Rest of the steps are identical to both of the ways of profiling.

After you have done with all the sessions that you want, close the little profiling window and optionally your application.

Congratulations! you have successfully performance-profiled your application.

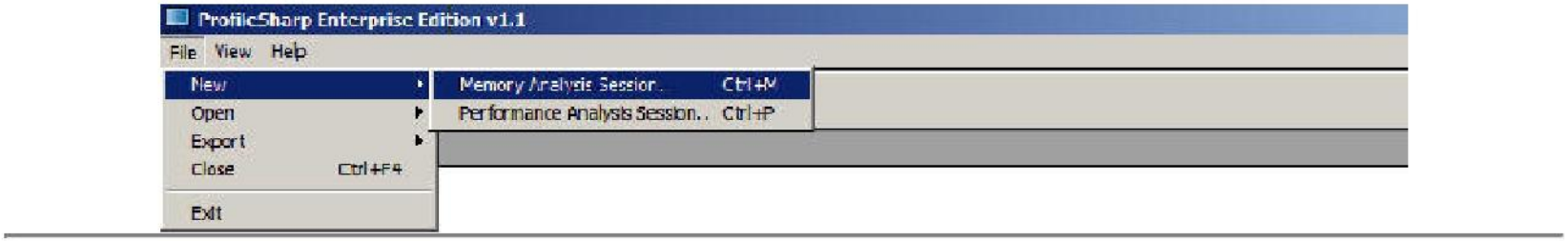
## Memory profiling. Steps to follow.

### Starting.

Memory profiling will help you collect information about objects in the .NET heap , their class names, age, allocating function with complete allocation-stack, their in-memory count and their size in bytes at any stage of your program execution, which you can analyze stand-alone or compare it with another set of information (a session) to figure out leaking objects, quantitatively and qualitatively.

To start a new memory profiling session , select the menu '***File->New->Memory Analysis Session***' or press Ctrl+M.

This will open the configuration dialog for memory profiling.



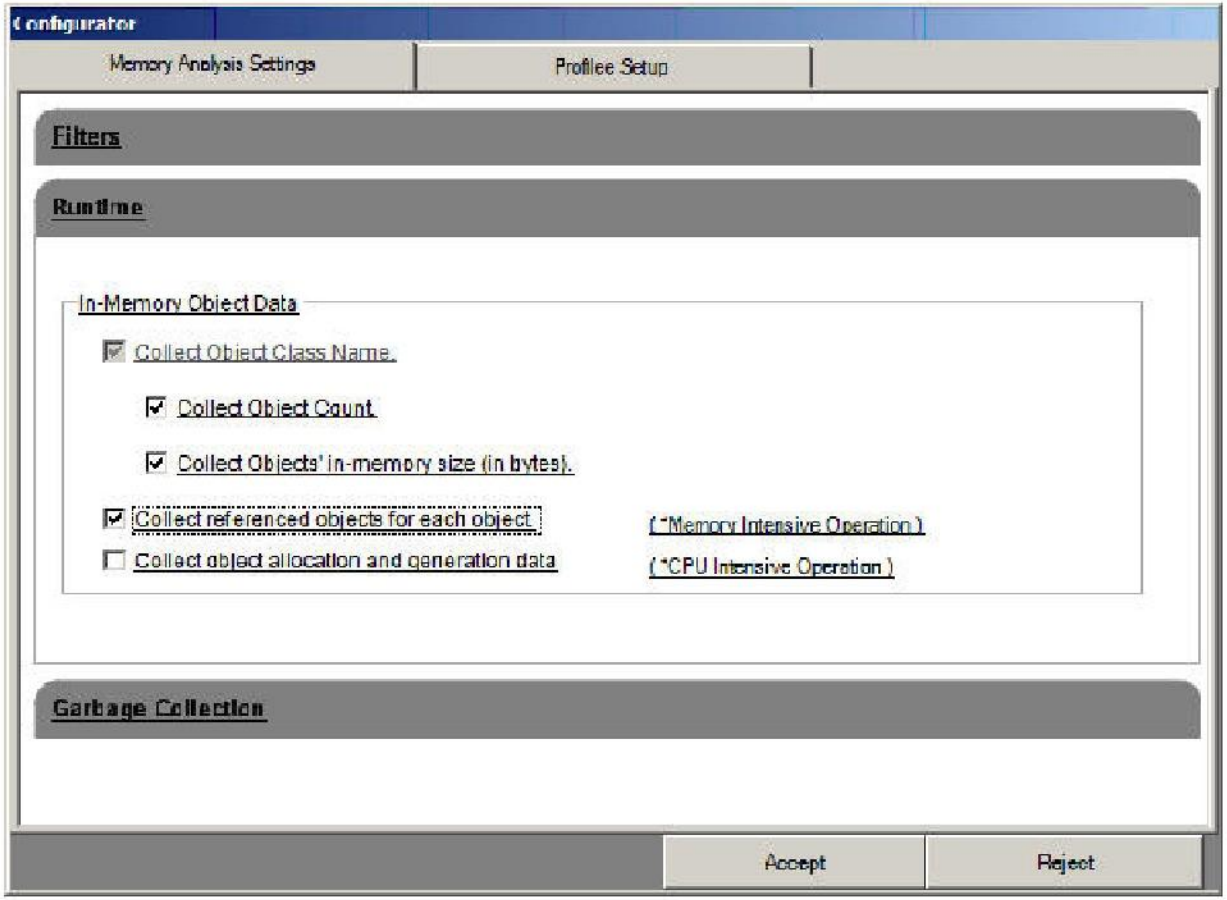

**Configuring the profiler. Choosing what you exactly want the profiler to profile.**

Once you start a new memory profiling session, you can set various options what to profile in your Profilee application, using the configuration dialog [the Configurator]. Click on the '**Filters**' sliding tab. The tab will slide-open. You can apply the filers to the profiling information you want to collect, same as you filter performance profiling information. Here you can however only apply namespace/class filter and no module filter. Filters always help you get to the piece of information that you want, blocking unnecessary and huge data, that may otherwise be collected needlessly. Also the profiler has to do lesser work. So you get more accurate results, faster. So applying filters smartly can invariably decrease your effort with the profiler.

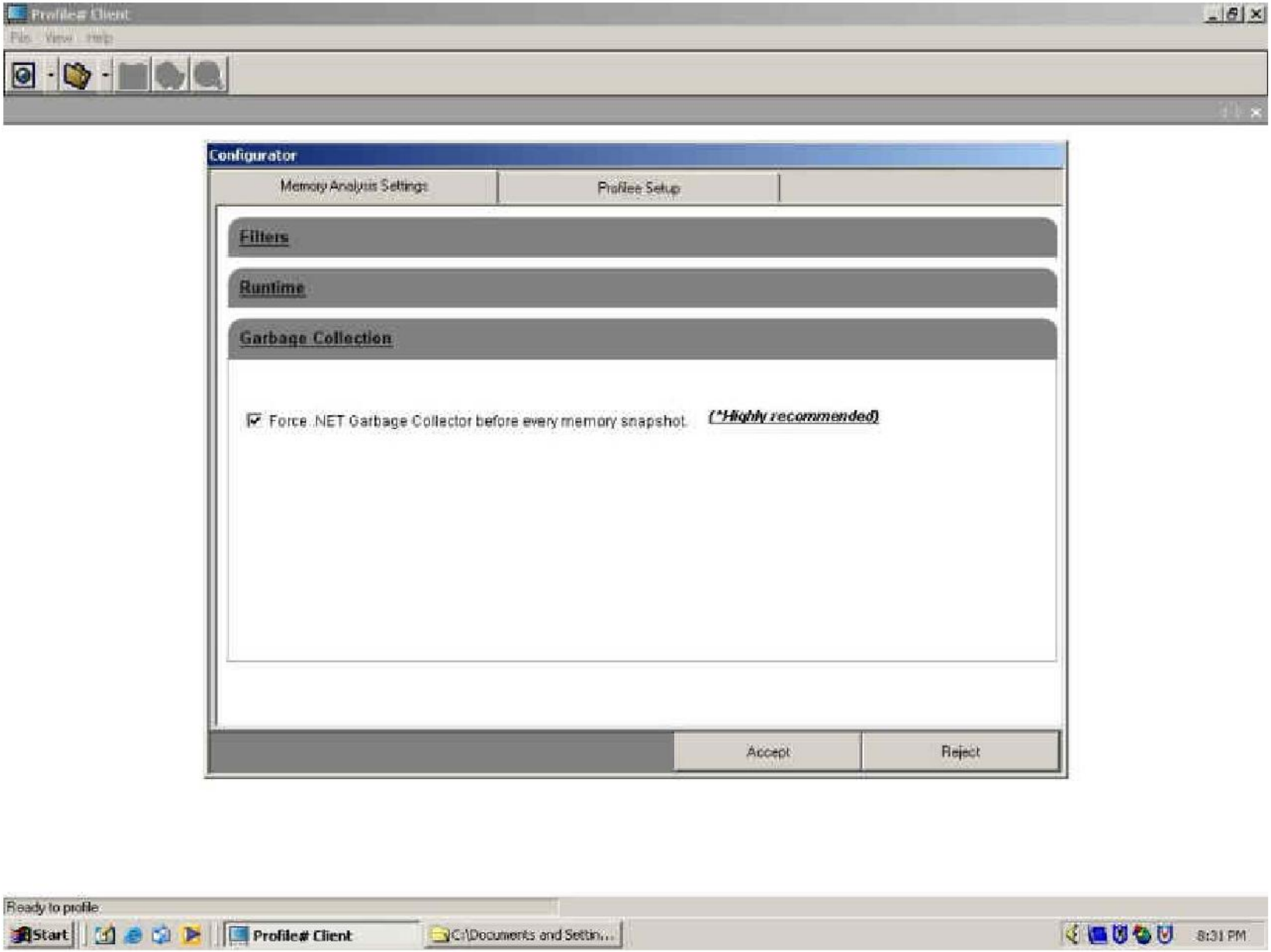
---

The **Runtime** tab is used to configure what actually the profiler will collect during memory-profiling. You can optionally select the option '**Collect referenced objects for each object**' to actually profile the details of the references held by each object to other objects in your application's .NET heap or let the default options. Selecting **object allocation and generation data** will enable the profiler to collect information regarding each object's allocation-stack as well as its **age.**
**Allocation-Stack** is the state of the function-stack in the thread that allocated the object, at the time the object was created.
**Age** is the number of .NET garbage collections, the object has witnessed since its inception.

**Note:-** Selecting the **object allocation and generation data** option is useful only if the profilee application is profiled from starting i.e. is right from beginning. Turning on profiling on an application after it has already started may not yield accurate allocation information for the objects collected. Hence it is strongly recommended that when you want the profiler to collect object's allocation and generation data, make sure you have started memory-profiling the process in its suspended state itself.
(i.e. choose option 2 or 3 of the **Profilee Setup** tab)




The **Garbage Collection** tab is for making sure that the garbage collector is run before every time a snap-shot / capturing of the .NET heap in your application is done. This cleans up disposable and collectable objects from the heap, which otherwise would appear as leaking objects. By letting this option checked, you are always left with only those objects that persist garbage collections so you can accurately analyze which objects should be and which should not be in the heap at a time. Also by comparing 2 memory-profiling sessions you can figure out the leaking objects and the leaking heap size to pin-point accuracy [as we will later see, how we can do this with ease]

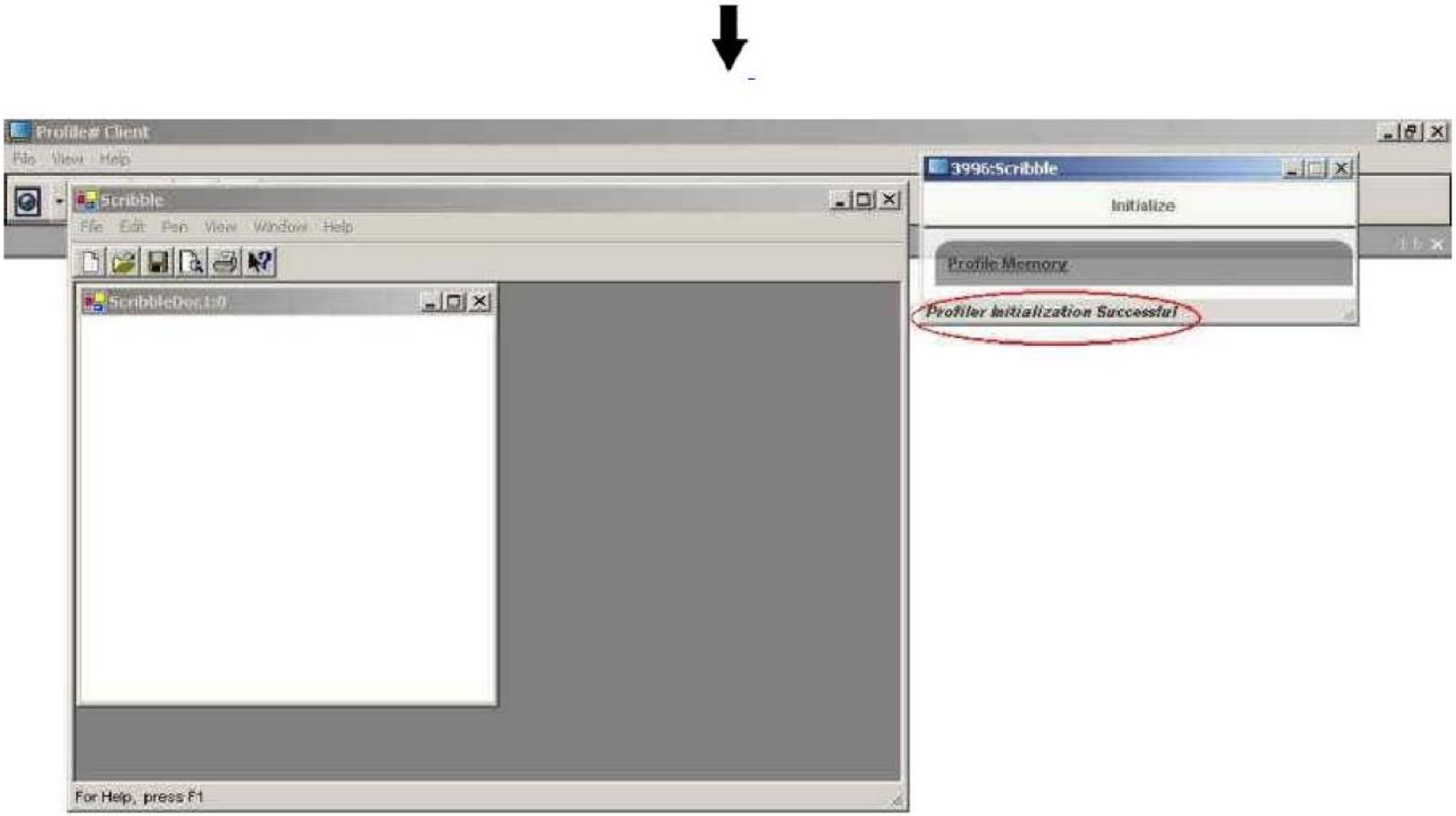





### Initializing the profiler

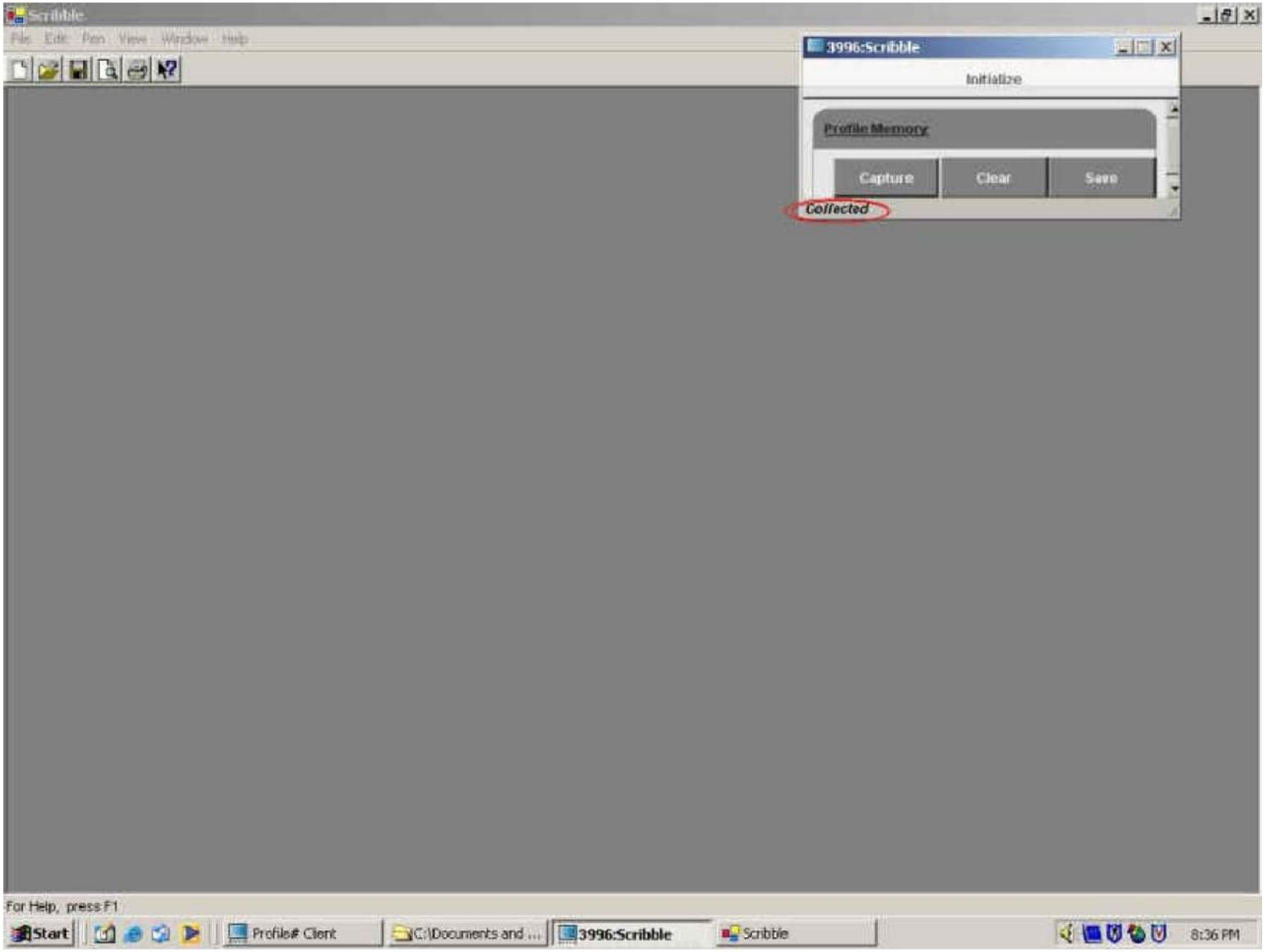
Once you have clicked the '**Accept**' button in the Configurator, your application will be started as normal. No profiling has been started as yet. Now

you will also see a small window on your screen, titled with a number and the name of the executable being profiled along with your application up and running (in case you had selected to memory profile your application once it has started). The number is the ProcessID of your application.
Click **'Initialize'** button on the window [This is always the first step you will do before profiling your application.] Clicking this button ensures that everything is fine and the profiling environment has been successfully loaded into your application, other wise it will popup an error message describing what caused the failure.




**Capturing state of the .NET heap. When and how to**

Now that initialization is successful, we can get on with memory profiling. Click the **Capture** button to immediately capture the exact state of the .NET heap ,the objects in memory, their individual size, references held etc. or you can first perform some sequence of operations / test-case in your application to lead your application to the point where you want to start profiling it and then press the '**Capture**' button. The status bar will show '***Collected***' when all the memory information of your application's current state has been collected. All this information at this point is now clubbed in a memory space separate from the .NET heap that your application is utilizing. Hence again the memory-profiling results that the profiler will capture and later on produce for analysis will be independent of the profiler's own memory consumption and so will accurately project the real-time scenario for your application.




**Saving the session.**

Press the '**Save**' button now to save the information to a session in the form of a file. This is most important as otherwise all the in-memory information may be lost if you proceed any further without actually saving the session. This is the same scenario as explained previously in context of performance profiling.

Please note again that you must save the profiling data to a directory that has *Write* permissions enabled for the user account, the *profilee* application/ process (and not the profiler) is running under. This scenario is most relevant in case you are profiling ASP.NET which by default runs under the account of the user ASPNET. The best thing here would be to save all the profiling sessions to a directory that has write permissions enabled for *Everyone*. Otherwise the session may not be saved giving an *'Access Denied'* Error!. Of course this issue is relevant only if you are saving the profiling sessions to a drive with NTFS file system. (See FAQ for more information)

---

### Clearing the cache. Closing down, and some things to remember.

Press **'Clear'** to clear the profiler of any lingering redundant information. At this point now, you are all ready to take another fresh snapshot of memory. A point that must not go unread is that all the profiling-sessions you save must bear unique names. Otherwise the profiler may not be able to distinguish between 2 distinct sessions later on when we analyze the sessions simultaneously [say for comparing them], giving duplicate or skewed results.

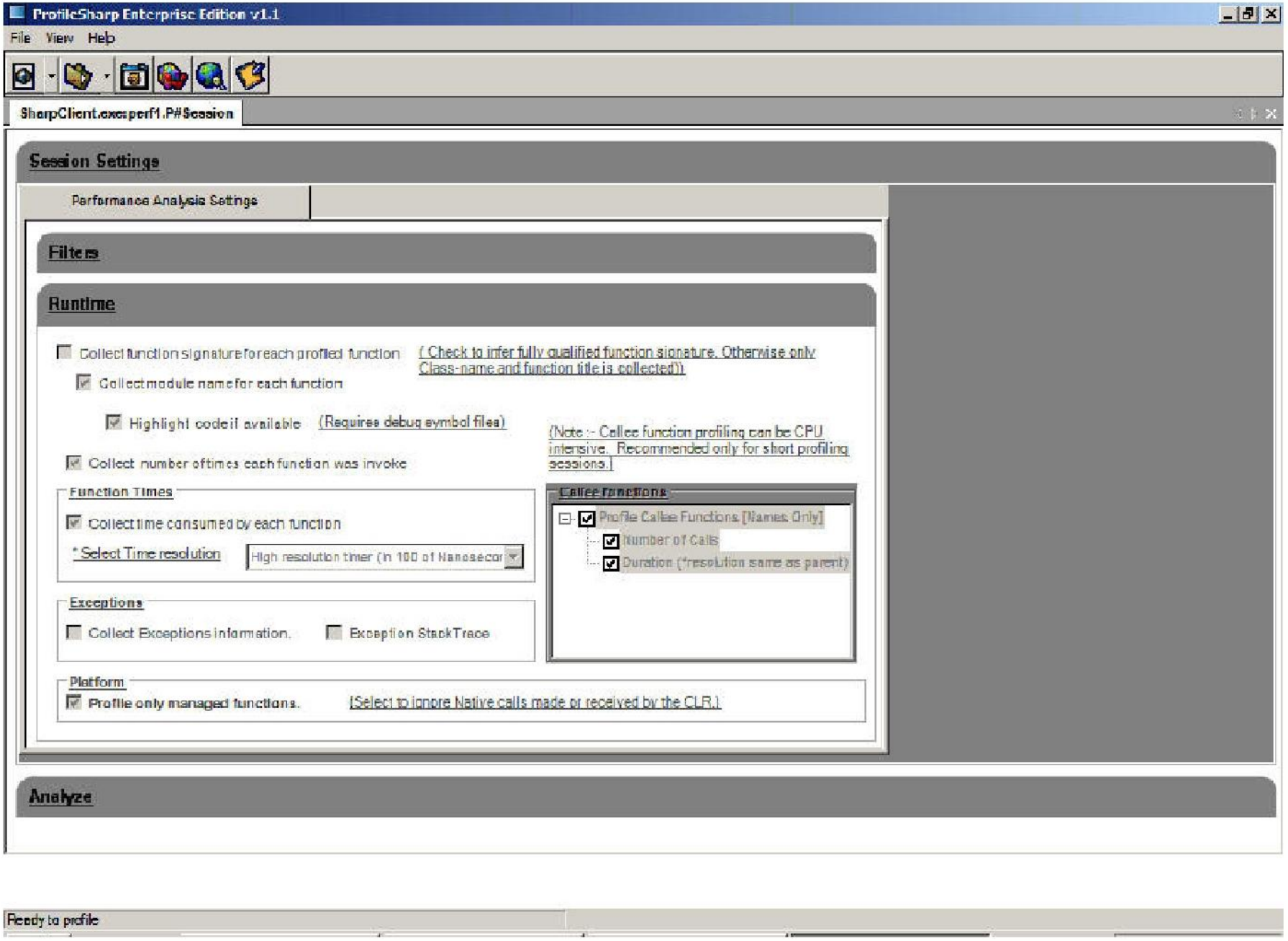
In the coming up pages of this document, we will also see how you can not just analyze the individual results of each profiling session but also compare 2 sessions in all sorts of different manners to dig-down to the actual number and size of objects leaking in your application. Note that unlike performance profiling , memory profiling actually captures the state of your program's .NET heap at a particular instance of your program. On the other hand performance profiling is more like , continuously monitoring and tracing your application for function calls over a time-period , spanning between **Start** button pressed, to the **Stop** button. That is actually how the two types of profiling paradigms [the performance and the memory profiling] differ inherently. However it is not uncommon to have memory profiling too actually span over a time, particularly to monitor which class-object was allocated by which part of your code over a time-period, or the life-time of an object in memory.
So once again, you are done with memory-profiling your application. Cheers!

## Performance Analysis [Steps to follow]

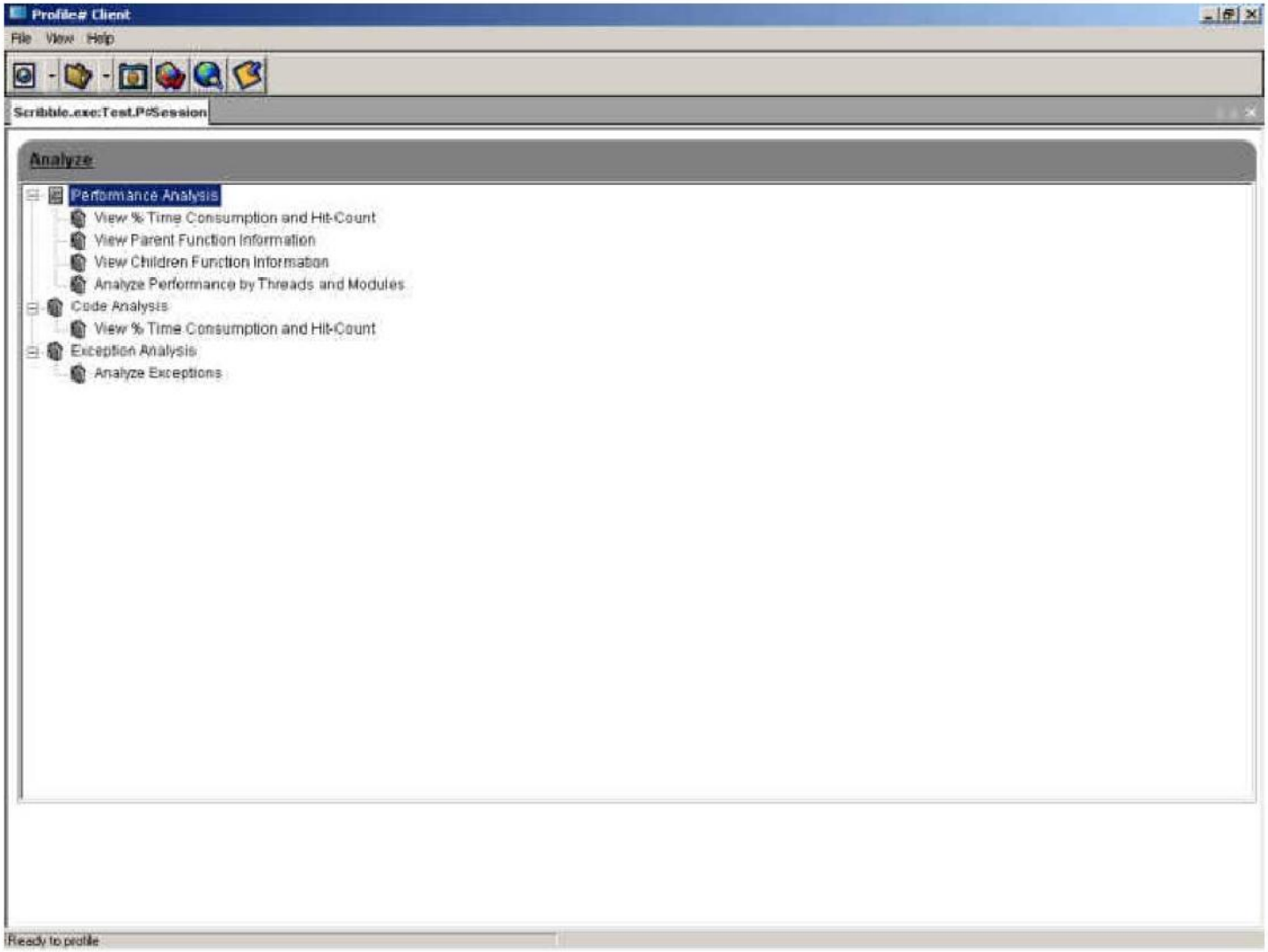
### Opening a saved session.

Now that we are done with collecting all the profiling information, we can proceed towards analyzing it in order to find what we want the profiler to...! Memory Leaks and Performance Hotspots. Taking on performance analysis first, we need to open a saved performance session first. So select the menu ' **File->Open->Performance Analysis Session**'. A file-open dialog will pop-up. Navigate to the location you saved your profiler sessions to. Select the appropriate session file [with extension ***.P#Session.fxml***] you want. This will start loading the session into the profiler in the background [in a worker thread, technically] .It may take a few seconds [may be a minute or two, in case you saved a session with no profiling filters or a very long profiling session]. You can always do some other sort of analysis or profiling in the meantime, as the profiler shall still be running normally. Once the session is loaded, you will see two active sliding tabs in the profiler's main window as depicted below. One with the '**Session Settings**' title and the other with '**Analyze**'.

To see what configuration settings you had chosen for this particular session, click the '**Session Settings**' tab . This tab will display the exact profiler settings for this particular session you had earlier chosen while profiling , [in read-only mode] so that you can recapitulate the context of this profiling session and not miss the details. You can toggle the visibility of this tab by pressing the '**Hide/Show Settings**' toolbar button above in the toolbar for the comfort of enlarging the analysis window.



Now click the '**Analyze**' sliding tab. The tab will slide-open, showing a list of neatly grouped views or projections of the collected profiling data. Choose an option that suits your requirement.

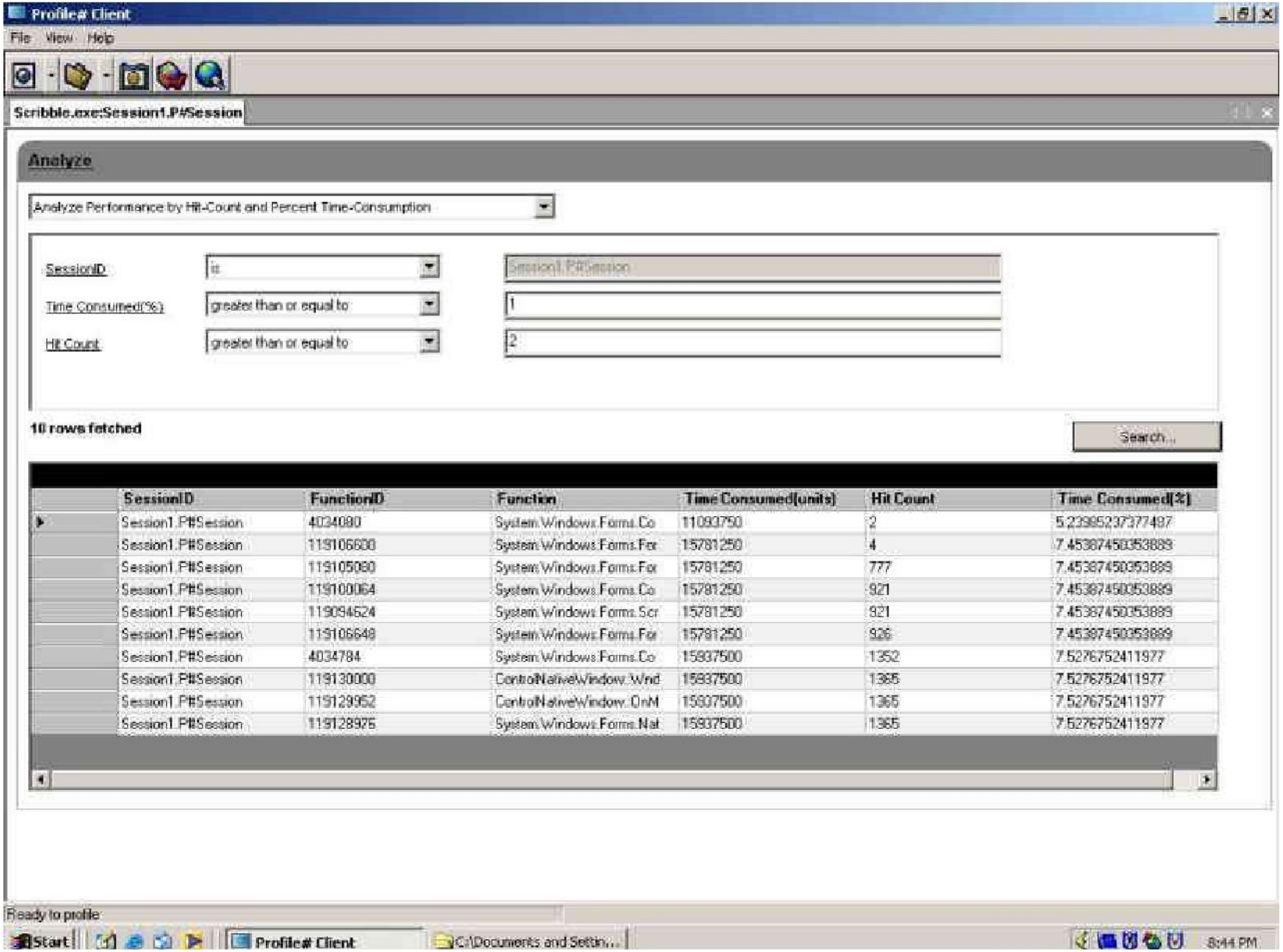



---

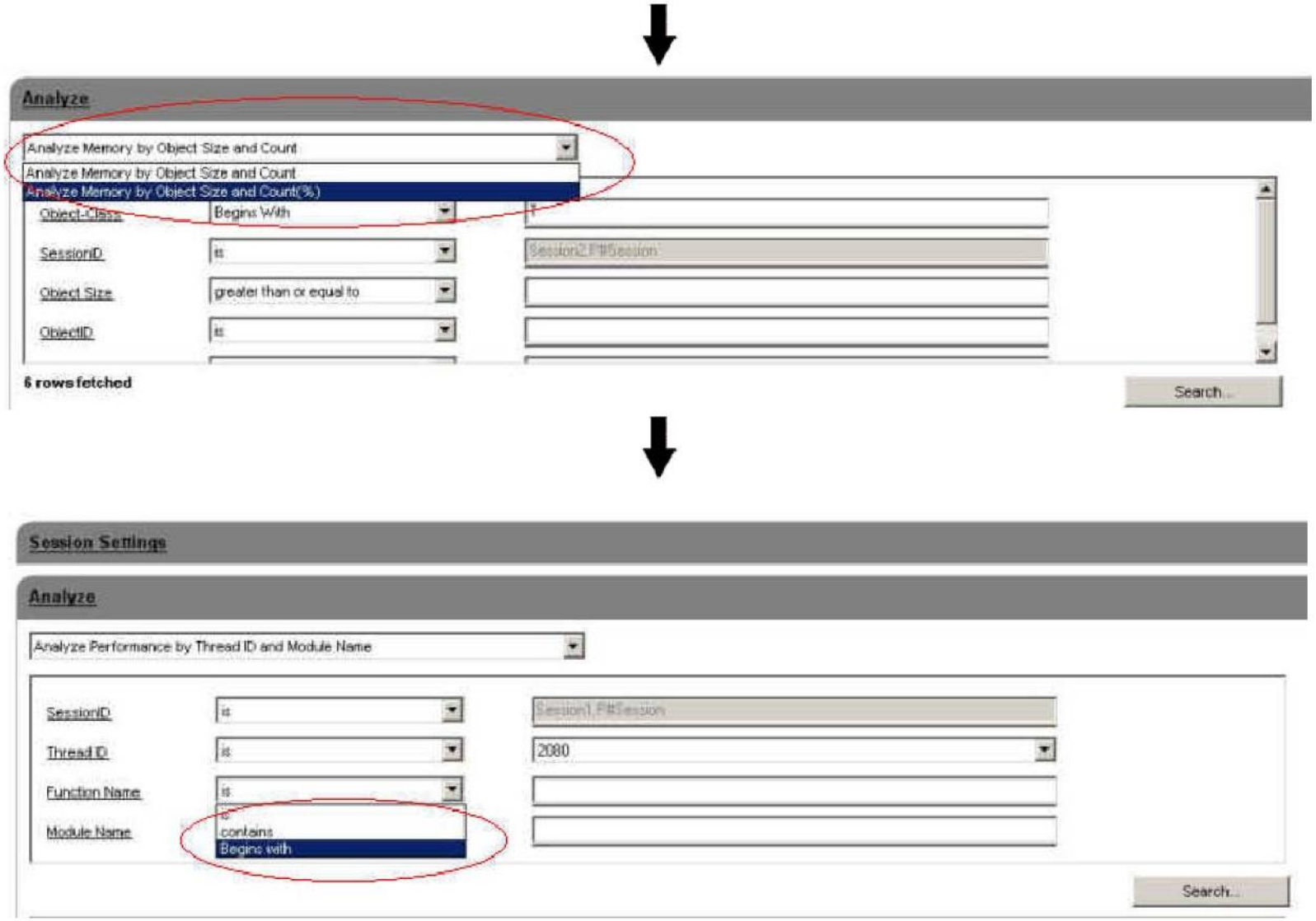
### Here goes the detail of each report :

I.) **View % Time Consumption and Hit-Count**. *Double-click* this list -item to load the view for analyzing the time consumed by each function as a percent of total time-span that your application ran for **this particular session**, or to view how many times each function was called. On selecting

this view , you will be presented with an analysis-tool called the '**Report-Grid'.** The report-grid is an integrated session analysis tool/ component that makes very convenient for you to filter out and navigate the profiling data in the way you like. It gives instant results and you can get down to each and every trace of information collected by the profiler. Just enter the criteria or scope of the information you want and click the **Search** button to view the results. The user-interface is easy and intuitive and is left with the user to explore. As an example, say you want to find out which functions during the particular session, were called over 5 times and consumed 3% of the total session time. So type in the values in the corresponding input fields and press **Search**. You will be presented with the details of all such functions, [FunctionID (a unique number for a function, only for data integrity purpose), time consumed(%),time-consumed units* ), Hit-Count , Function Name etc.].



Alternatively you can leave all the fields blank to get an exhaustive search for all the functions called during the session. You can also change the **searching operator expressions** [ the drop-down combo-boxes just before the input fields or at the top, just below the title of the sliding tab, shown in red in the 2 pictures below ] , provided there are more options available for that view.

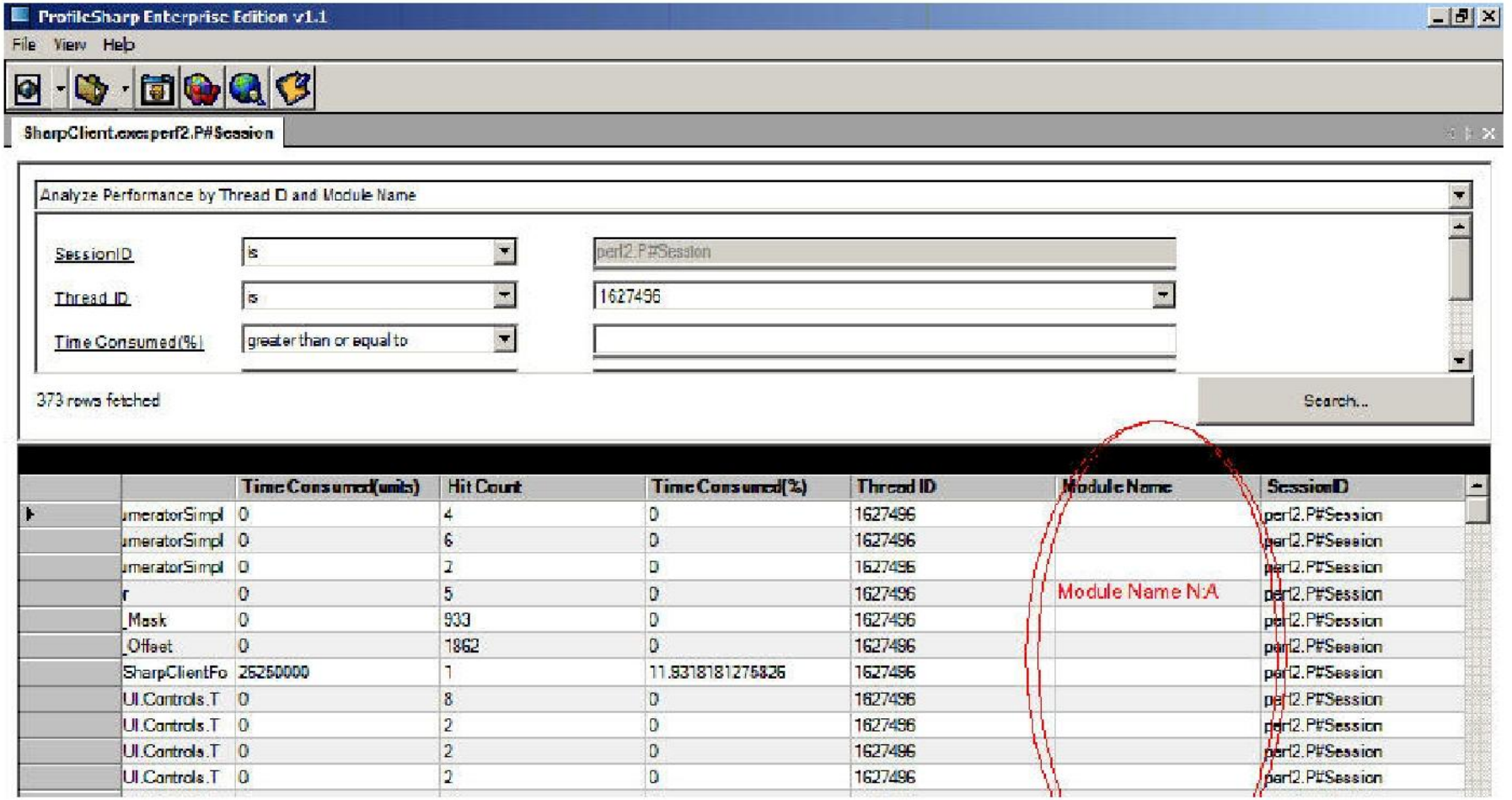



The same fashion of filtering the data will apply to all the views and reports that you choose

II.) **Analyze Performance by Thread and Modules.** This view , obviously helps you search for functions and their details [hit-count, time consumed etc] on the basis of the thread-id the functions were executed upon or the module names the function belongs to. The technique to filter and hunt-down the required functions is same as explained above in the first point. Some other points to be clarified here are :-

a.) In this view you can see how the **searching operator expressions** [the drop-down combo boxes marked in red above ] can be adjusted to expose different information searching and analysis modes of the ReportGrid. What this means will be very evident just now. Just click the combo box next to '**Function Name**' label in the ReportGrid and select the option '**Begins With**'. With this option selected , you can now input and subsequently filter-out the functions that begin with a particular letter or string **prefix**, rather than typing in the complete function name as is required by default.

b.) Another valid point to be stressed here is that you may sometimes see **no** information for some of the fields [and columns] during analysis. This is because you had not selected those parts of information to be collected in the **Configurator** before profiling the session. An example of this is that you may see no module name drop-down option in the field next to the **searching operator expression** for **Module Name**. Nor will the results presented by the ReportGrid contain any information about the same. This will happen in case you had not configured the profiler to collect module name information during profiling.

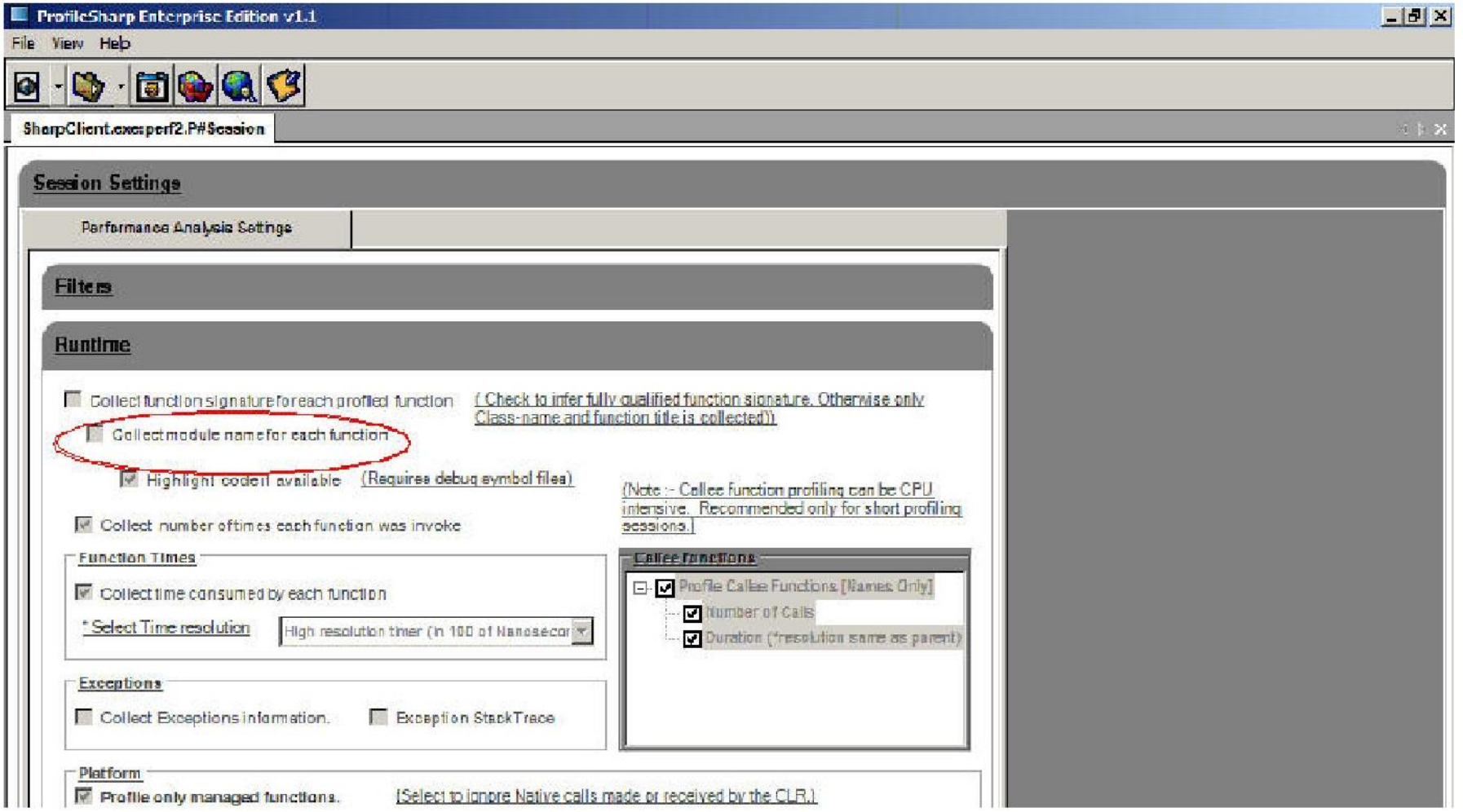


| | | | | | | |
|---|---|---|---|---|---|---|
| | UI.Controls.T | 0 | 2 | 0 | 1627496 | perf2.P#Session |
| | UI.Controls.T | 0 | 2 | 0 | 1627496 | perf2.P#Session |
| | UI.Controls.T | 0 | 2 | 0 | 1627496 | perf2.P#Session |
| | UI.Controls.T | 0 | 3 | 0 | 1627496 | perf2.P#Session |
| | UI.Controls.T | 0 | 3 | 0 | 1627496 | perf2.P#Session |
| | UI.Controls.T | 0 | 4 | 0 | 1627496 | perf2.P#Session |
| | UI.Controls.T | 0 | 25 | 0 | 1627496 | perf2.P#Session |
| | UI.Controls.T | 0 | 1 | 0 | 1627496 | perf2.P#Session |
| | UI.Controls.T | 0 | 1 | 0 | 1627496 | perf2.P#Session |

Ready to profile

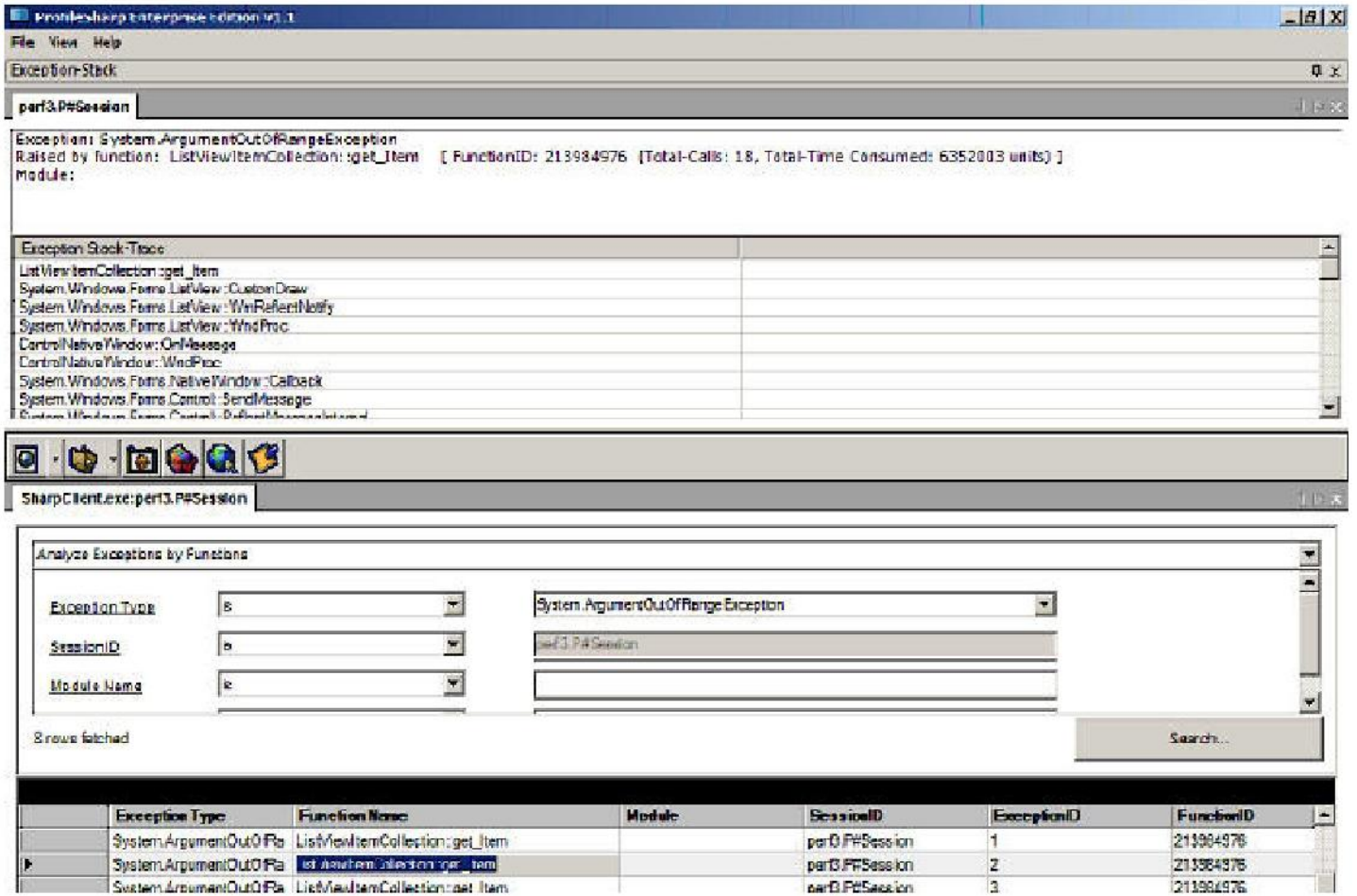
Start | Microsoft FrontPage - ... | SharpClient | SharpClient - Microsoft... | ProfileSharp Enterp... | untitled - Paint | 10:18 AM

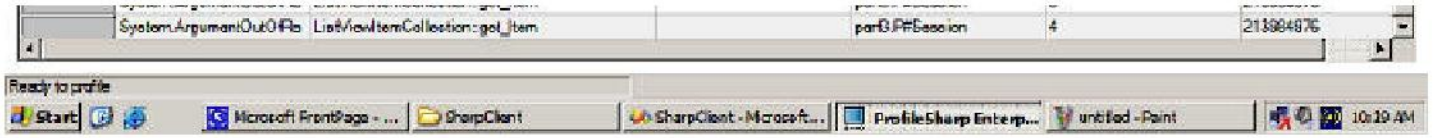
You can confirm this by viewing the '**Sessions Settings**' sliding tab above (see below for a picture). All other fields and columns follow the same pattern.





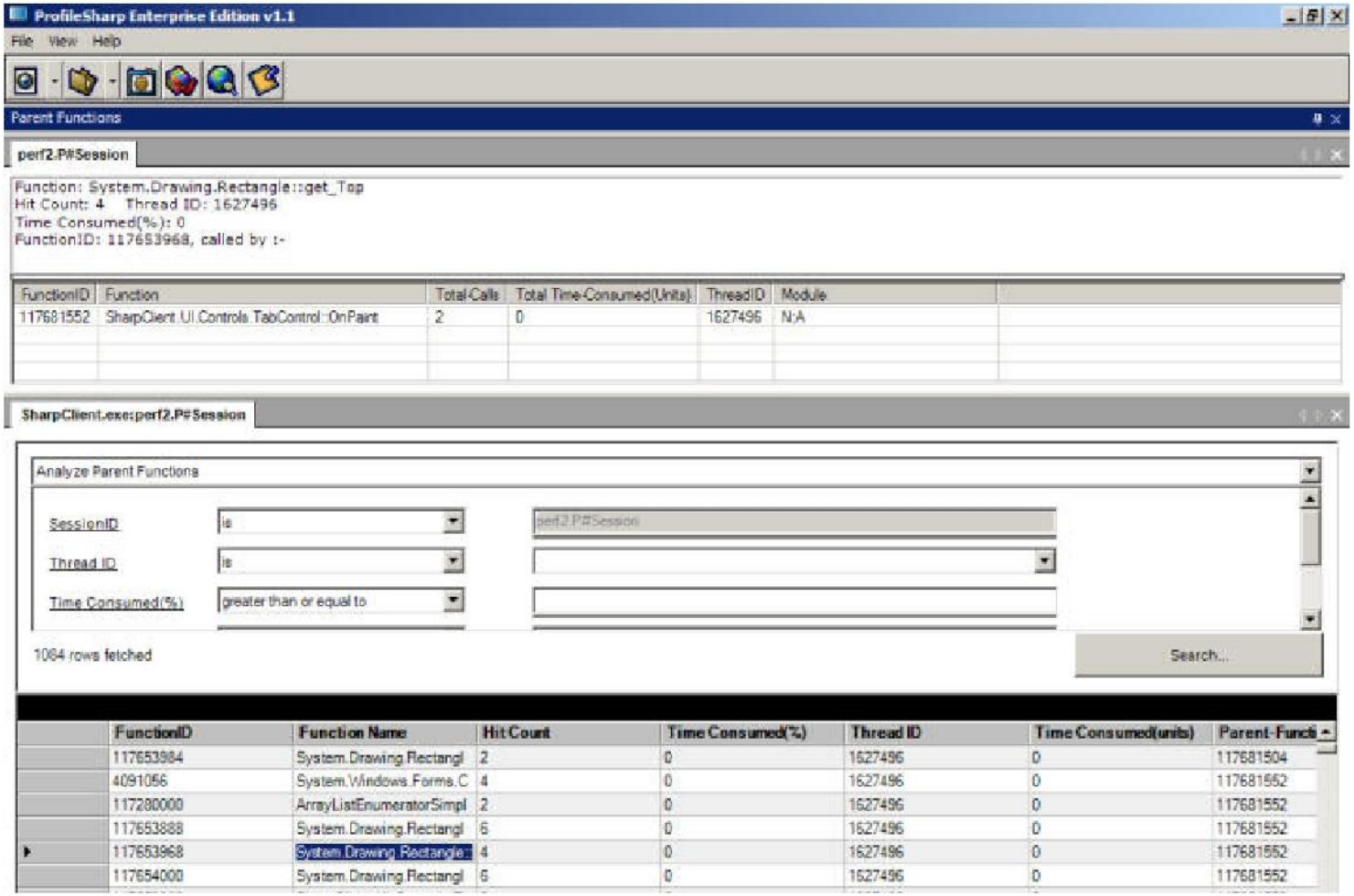
**III.) Analyze exceptions** .Select this list / report to view the exceptions that were raised [caught or uncaught**] while the session was in progress.
You can also in fact see the detailed stack trace when the exception occurred. Click on a row to view the stack-trace for each exception raised [only available if ' Collect Exception-Stack Trace' option was selected for the session before profiling in the Configurator]. This will bring forward the **Exception-Stack** panel with exception-trace for each exception raised.

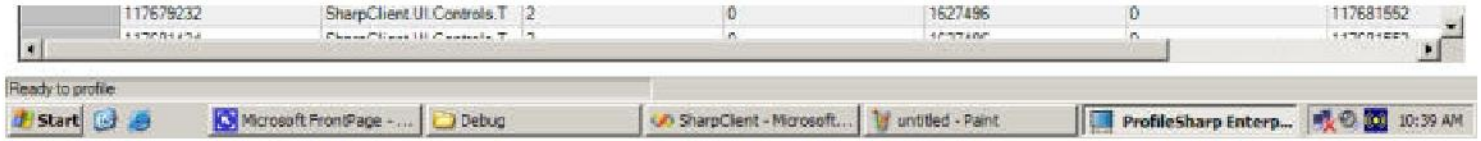




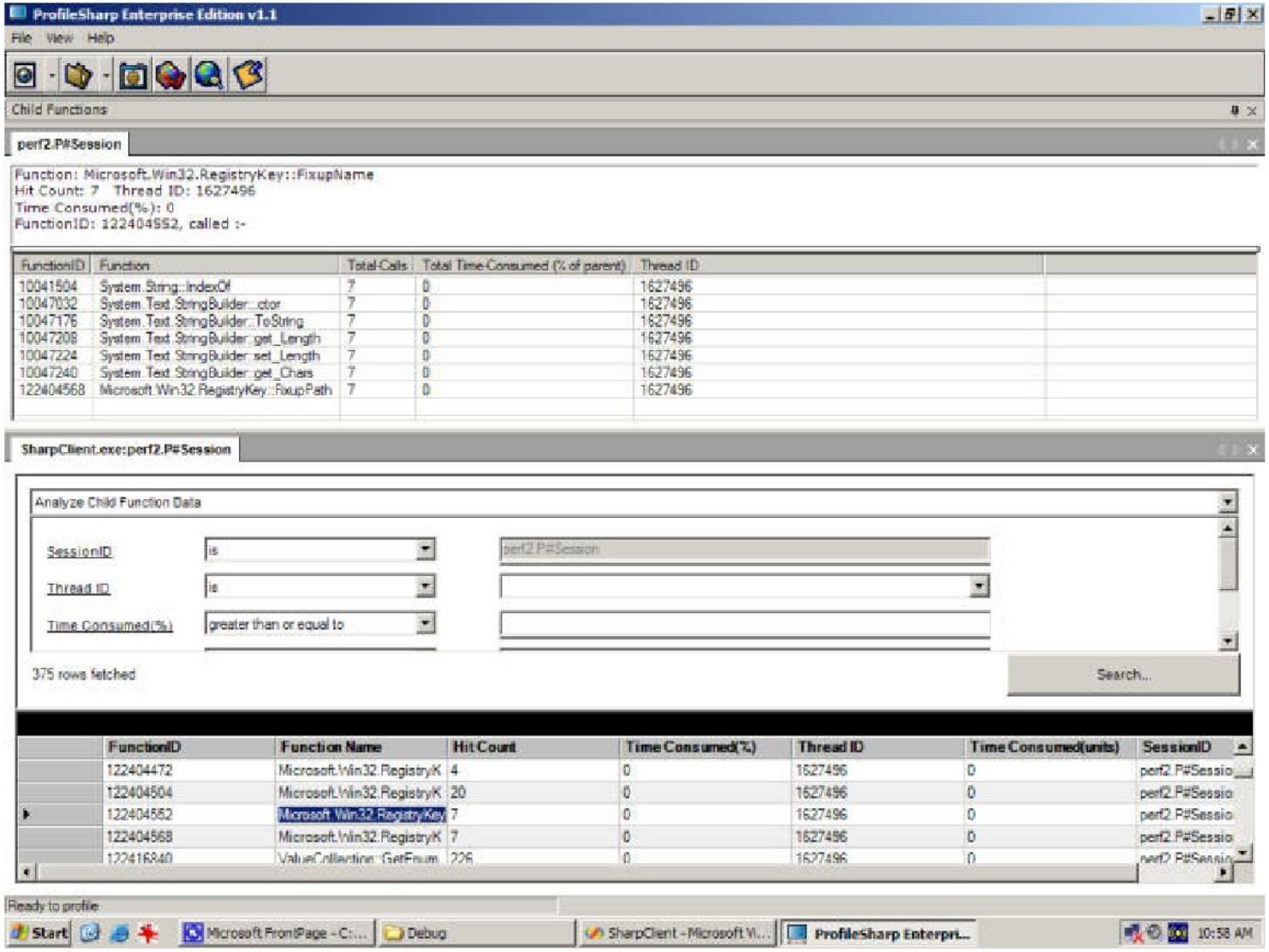

You can always apply more than one views to a particular session for analysis purpose simultaneously, as the ProfileSharp is an MDI application. Only for the first time will the loading of a particular session for analysis take some time. Successive analysis views when applied will be directly opened on already loaded and cached session data. To open another view just follow the same steps as you did for the first view . Later on you can choose a different Report/View to be applied to it.

**IV.) View Parent Function Information.** Select this report to analyze the parent functions for each function called i.e. the functions called and the functions that called it (parents). Note that in case a function was called by multiple functions, it will have more than 1 (corresponding) entries for each of the time it was called, each entry containing the detail of the parent function that called it. You can navigate through the parent function of each function by clicking on the corresponding row in the result grid.






**V.) View Children Function Information.** Select this report to view the children functions for each function i.e. all the functions and their details which were called by a particular function. In case of a function having multiple children (i.e. it called more than 1 function during its execution), it shall have multiple entries in its '**Child Functions**' panel, each corresponding to the function it called. You can navigate through the child functions of each function by clicking on the corresponding row in the result grid.

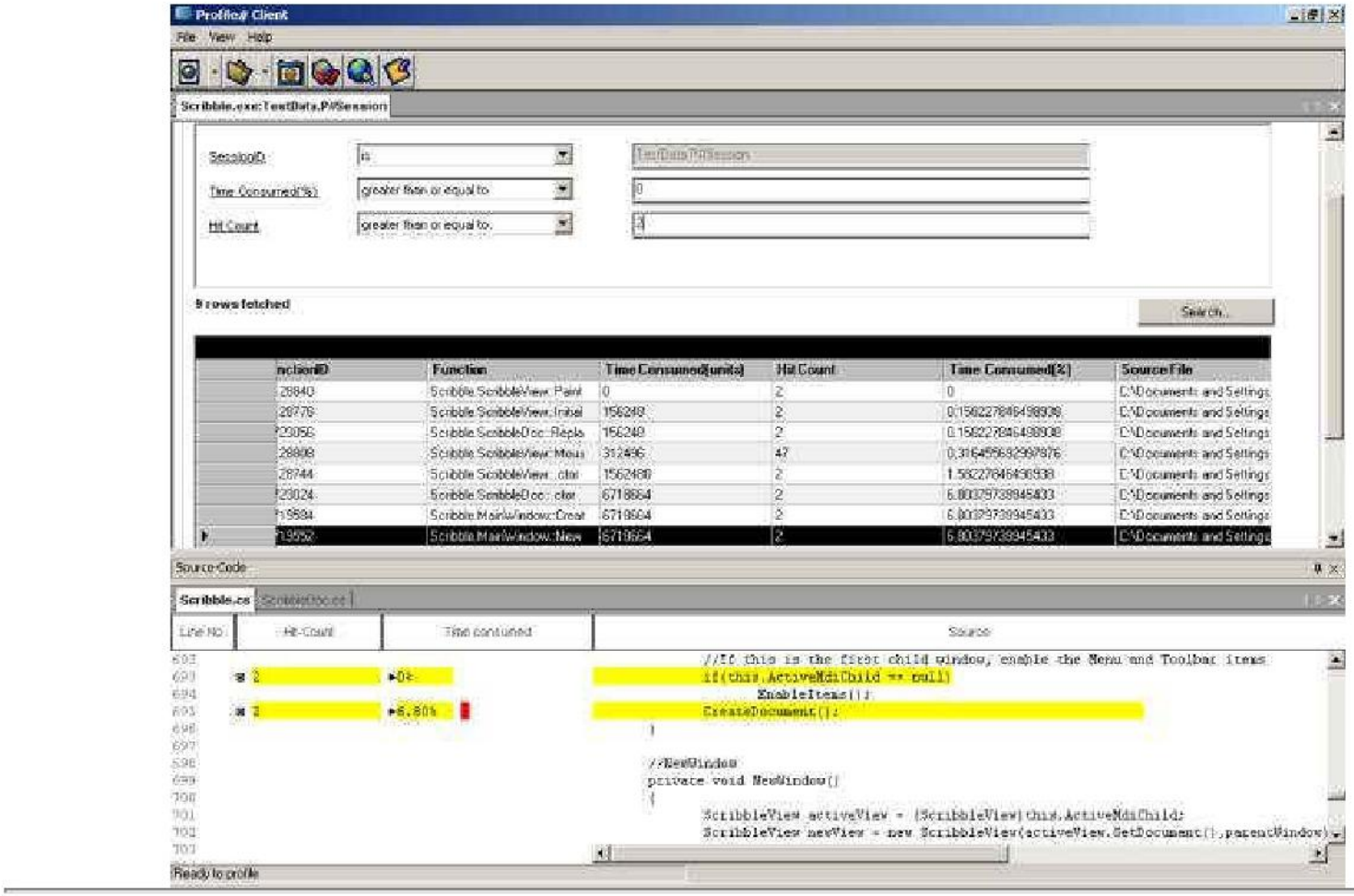


**VI.) View % Time Consumption and Hit-Count by Code.**

Select this option if you had selected to code profile your .NET application in the **Configurator** ('*Highlight code if available*'**)**. Selecting this report will let you analyze how your .NET code performed line-by-line.

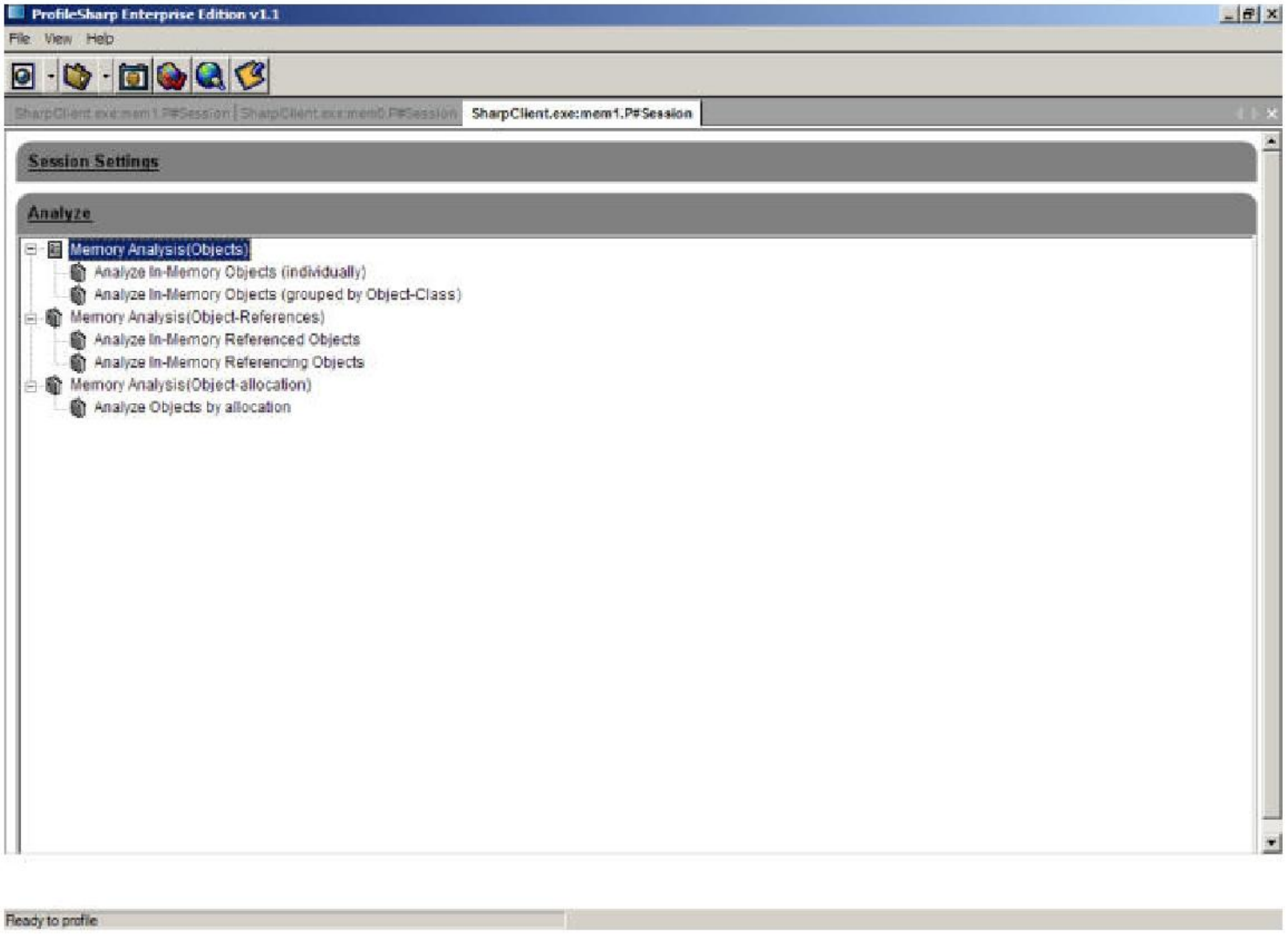
Once you have entered your criteria in the search-box and report-grid has presented you with the results, go ahead and click on one of the result rows with the method you want to see the code for. This will present you with a collapsible panel window at the bottom of the Profiler's main window containing the corresponding code, highlighting the lines along with its details like 'Hit-Count' and '% Time Consumed' [*The **Time Consumed** field shows the time consumed by each executed line of code, as a percent of the time consumed by the **function** containing it.*]

Click through the methods that you want to see the code-performance for and the corresponding source-code will be automatically loaded and presented to you simultaneously in the **Source-Code** panel. Code with higher % time consumption will be marked with a red bar so that you can visually figure it out.



Memory Analysis [Steps to follow]

Applying the same analysis pattern to Memory Analysis as discussed above for performance profiling, select the menu '**File->Open->Memory Analysis Session**'. Browse to a saved memory session file [a file with extension **P#Session.oxml]**, and open it. Choose and apply the appropriate filter/view to the loaded session.




Here are the filters available :

**I) Analyze In-Memory Objects (individually)**. Apply this view to visualize the Object ID [a unique number assigned to each object for reference purpose] ,size [in bytes] , size as a percent of the total size of all the objects, class name of **each** object , object count expressed as a % of total object

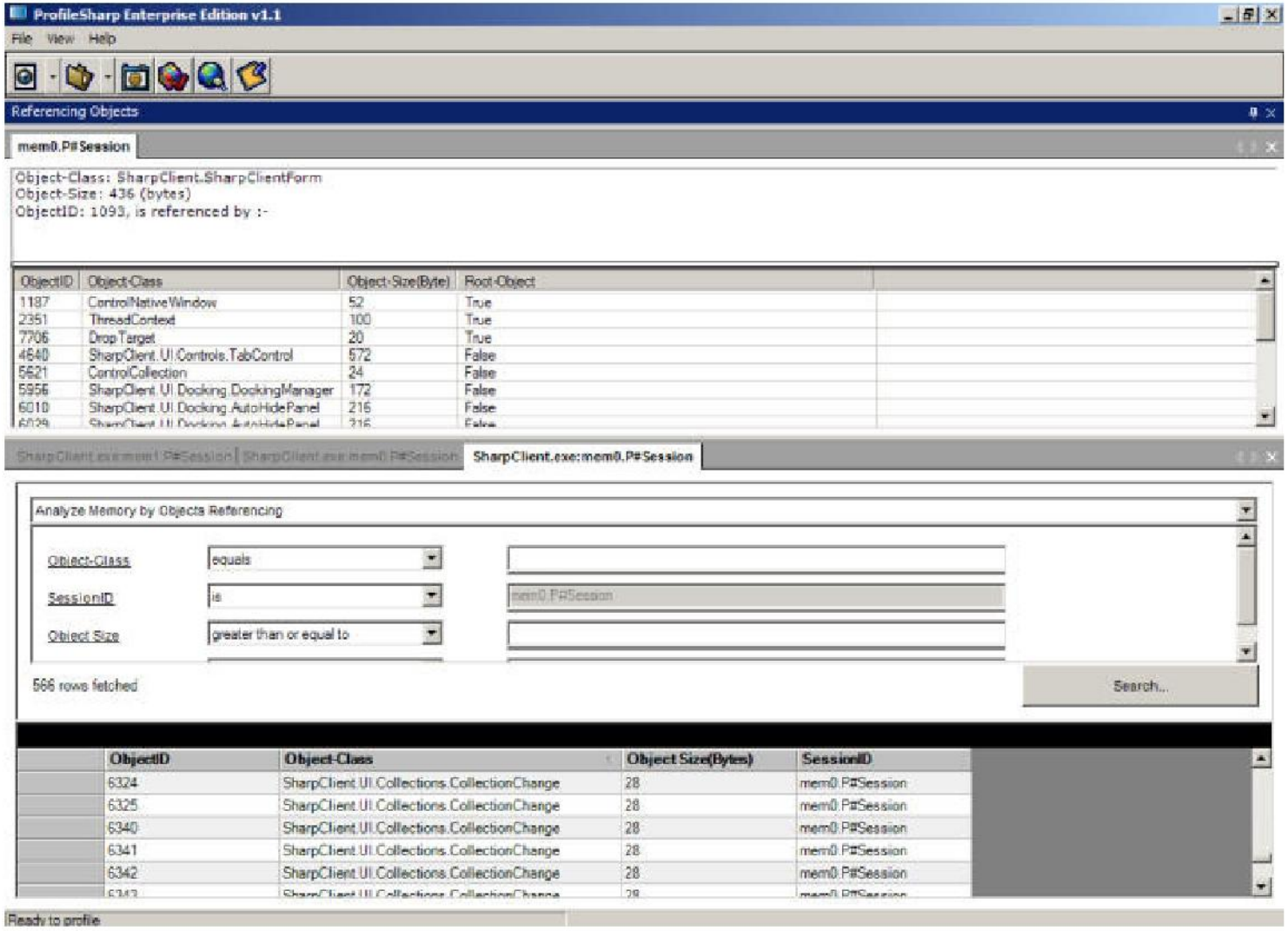
count and whether the object is a 'Root Object' (a 'Root Object' is one which is at the top of an object allocation stack).

**II) Analyze In-Memory Objects (grouped by class).** Apply this view to get the memory profiling results grouped by class-names rather than individual objects.

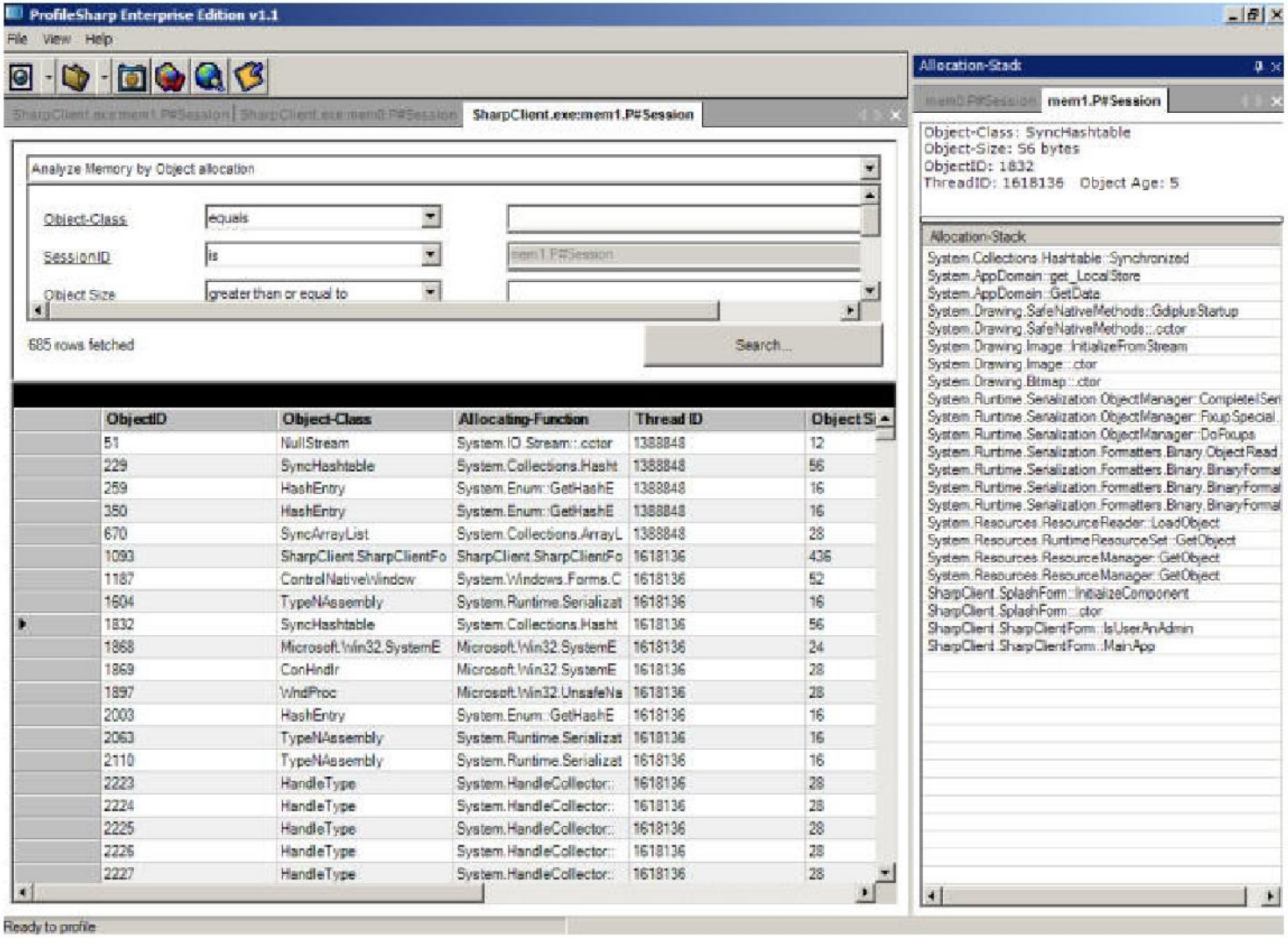
**III) Analyze In-Memory Referenced Objects .** Apply this view to analyze a particular object, its individual details (object size (in bytes), whether root-object or not ) along with the details of all the **objects being referenced by it** and the number of references the object is holding towards each of the referenced object (i.e. the Reference-Count). Click on a row to view the objects referenced [only available if '**Collect referenced objects for each object**' option was selected for the session before profiling in the Configurator]

**IV) Analyze In-Memory Referencing Objects .** Apply this view to analyze each object by the the references that other **objects are holding towards it** (Referencing it). Say an object *SharpClient.SharpClientForm* is being referenced by objects *ControlNativeWindow*, *ControlCollection* and *ThreadContext*, then the result pane for this scenario would produce 3 individual entries for the object *SharpClient.SharpClientForm* each giving the details of the object itself (*SharpClient.SharpClientForm)* along with the Reference-Count (number of references *ControlNativeWindow, ControlCollection and ThreadContext* are holding towards *SharpClient.SharpClientForm*, respectively).

Click on a row to view the objects referencing [only available if '**Collect referenced objects for each object**' option was selected for the session before profiling in the Configurator]




**V) Analyze Objects by allocation.** Apply this view to analyze each object's allocation patterns. This information includes the thread-Id of the thread it was created by, the function that released the object along with complete stack-trace that led to the object's creation, as well as its **age**. Age of an object is the number of Garbage Collections the object has/had seen during its lifespan. This information is available only if '**Collect object allocation and generation data**' option was selected for the session before profiling in the Configurator.




**Note:-** Selecting the **object allocation and generation data** option is useful only if the profilee application is profiled from starting i.e. right from beginning. Turning on profiling on an application after it has already started may not yield accurate allocation information for the objects collected. Hence it is strongly recommended that when you want the profiler to collect object's allocation and generation data, make sure you have started memory-profiling, the process in its suspended state itself.
(i.e. choose option 2 or 3 of the **Profilee Setup** tab)

*[units here represent the unit of time-resolution chosen to measure the time consumed by each function by you before profiling this session. Click the 'Session-Settings' sliding tab ,(Click 'Hide/Show Settings' toolbar button first if it is invisible). Click 'Runtime' tab and view the 'Function Times' setting. This is the unit that was chosen.] **[Uncaught exceptions will only be available for analysis if the session was saved before you chose to

close the Profilee program after the unhandled exception occurred. In future versions of ProfileSharp, you will not need to do this, as automatic saving of the last session will be done in case of abnormal termination of the application ]

Comparing Memory Sessions

### A word before we begin :

You have the profiling data, and you can now analyze it using the ReportGrid. But what is missing is comparative analysis. Unless you can compare two sessions, you can not deduce, what is the number of bytes leaking in your application and where. So to take you to this quantitative comparison results, ProfileSharp has got another essential offered feature.

**Memory Comparison Session.**

Earlier versions of ProfileSharp offered comparing Performance Sessions besides the normal Memory Comparison Sessions. However Performance Comparison feature has now been eliminated as it was an overhead in the profiler because performance analysis of an individual performance session is sufficient enough to give exhaustive information about the performance of the profilee application. Performance profiling results are self-explanatory (You don't need to compare a performance session to another, to deduce that a function in your application is taking 10% of the total time when it should only take 1%. The first session itself will have that information.)

You can open a memory comparison session by selecting the menu '**File->Open->Memory Comparison Session**' or press Ctrl+Shift+C. But like anything else , to compare something to other, you must have a contender. Equally important is the fact that 2 things can be compared objectively, only if both of them are bound by exactly same rules and criteria. What this means in context of profiling and **ProfileSharp** is that, at least 2 memory sessions must be already open for them to be compared in the profiler. If you have only one memory session open for analysis in the profiler, the comparison can simply not be made. Another key factor for proper and result-oriented comparison between 2 or more sessions is, that the sessions that are to be compared should essentially have had been collected on the same profilee application/process **without the target profilee application being restarted** between the 2 sessions we are going to compare. Also obviously, you can not compare the data of 2 sessions meaningfully, if one session holds the profiling data for your application, say, while it was connecting to a database and fetching some records while the other session contains the profiling data for you application printing some reports. So keep that in mind. We will show you how. Read On.

Taking up the task of explaining comparison sessions in context of Memory Profiling, and how to use it correctly to pinpoint the leaking objects. For this, we will explain step by step procedure using a common .NET application, the *Scribble* , a sample C# MDI application that ships with VS.NET. We will find out, how many and which class-objects remain in the memory (i.e. leak) along with the detail information for each object, when you open a new child form in the *Scribble* .Follow the steps :

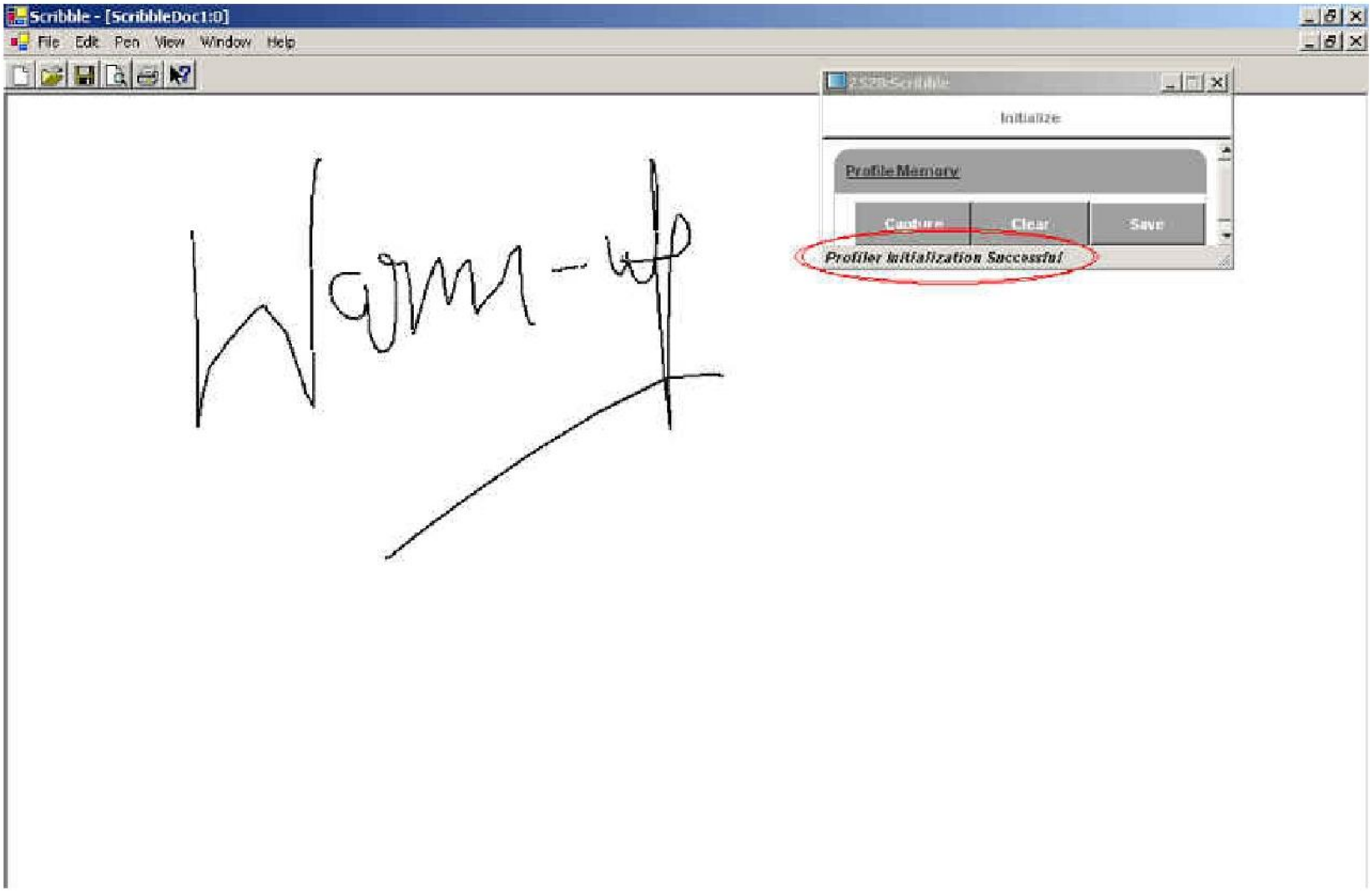
1.) Start the profiler, and select the menu option '**File->New->Memory Analysis Session**'. Let all the settings be **selected** in the **Runtime** tab of the configuration dialog and just move on to '**Profilee Setup**' tab. Browse to the folder containing the *Scribble* Application and select it. Select option 2

(**Start a new process and start profiling it right away** as we also want to analyze each objects 'allocation and generation' data ) Press the **Accept** button. You will see the Memory-Profiling Control's window.

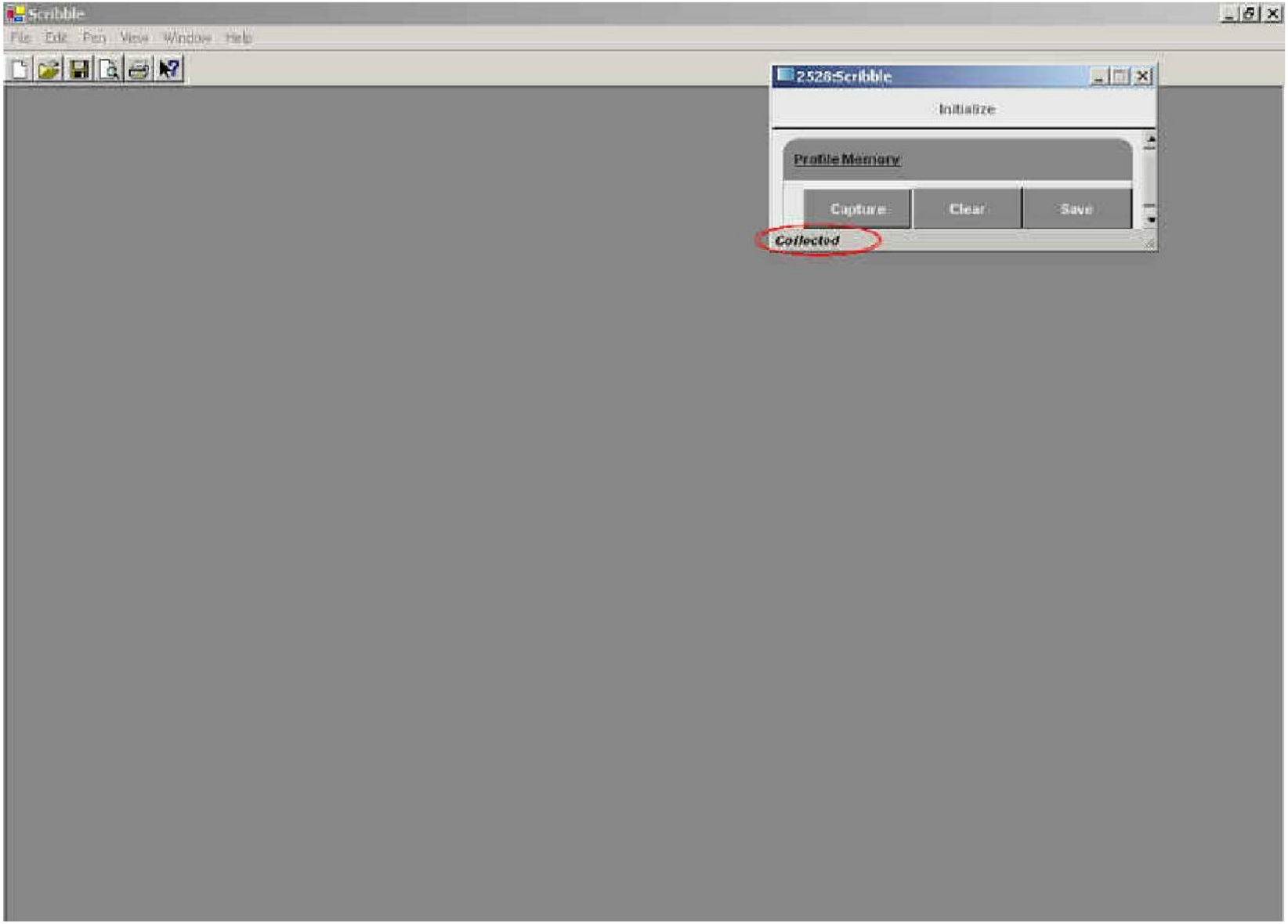
2.) Click the '**Initialize**' button on the profiler control's window. Verify that the status-bar of the profiler control shows '***Profiler Initialization Successful***'

3.) Now click the **Resume** button to resume the process.

4.) Do not click the **Capture** button now. Lets first warm up the application to a specific state where on from we can get the most appropriate data. So create a new MDI child window. Drag the mouse on it and draw some text. Now close the window. At this point the scribble application is ready to be memory-profiled for the objects created when you create a new MDI child window. Note that you should always first execute a warm-up flow for the test-case that you are going to profile, before you have actually started profiling. What this does is that it brings the application to a state where it contains all the necessary objects and memory resources which may be required by the test-case to begin but are not a part of the test case itself.

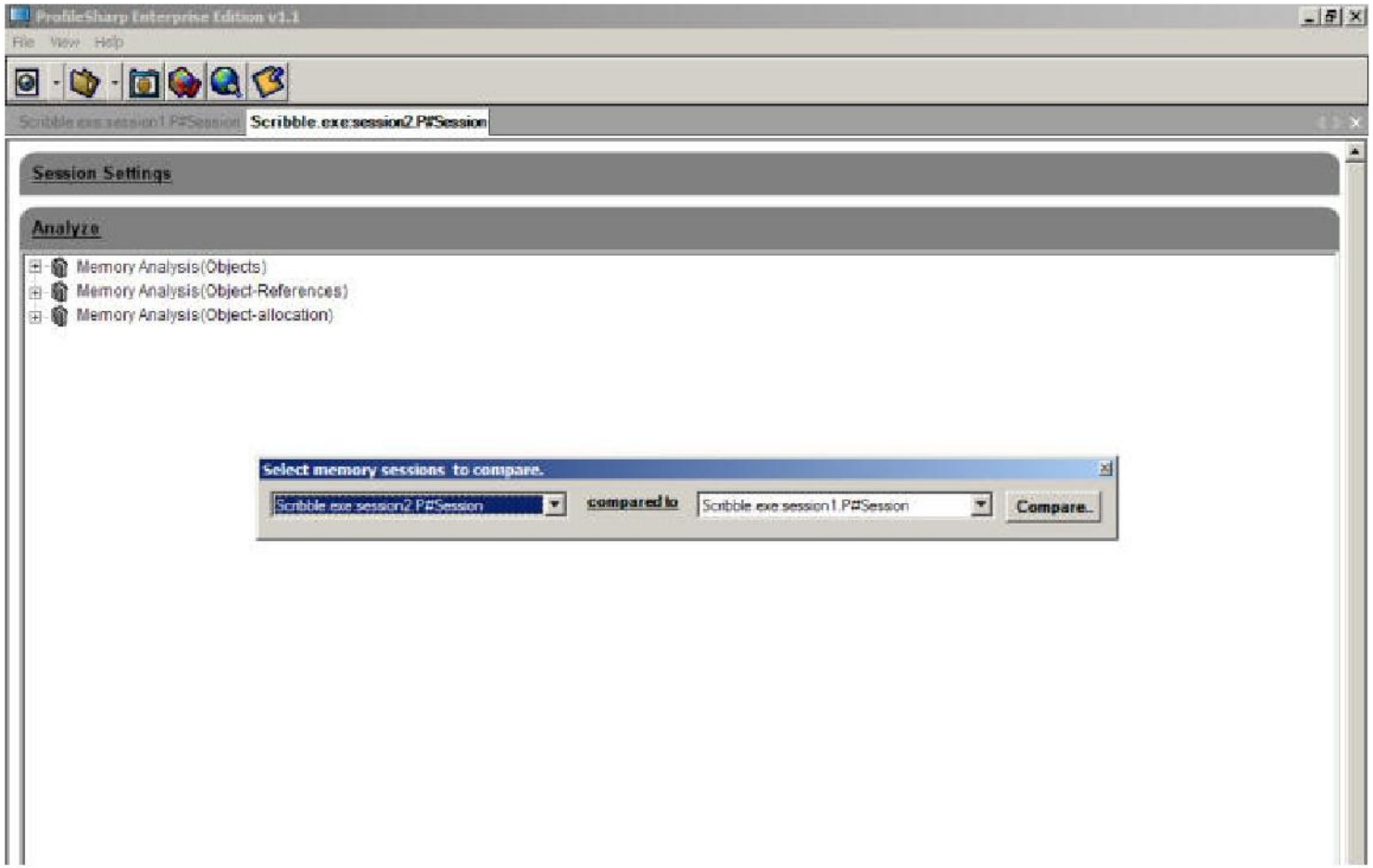


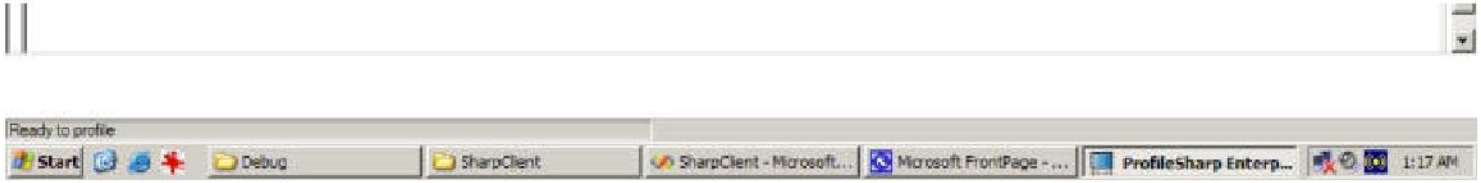

5.) Now let's start. Create a new MDI child and draw some text on it. Now close the child window. Click the '**Capture**' button to take a snapshot of the .NET heap of the application [Wait for the status-bar to show '**Collected**' as depicted below]. Press '**Save**' to save the session with a name say *Session1*. Click '**Clear**' to clear the profiler's memory from all the data for the session it has just collected.




6.) Again create a new MDI child and draw some text on it. Close the child window. Now we are at the same stage as when we collected *Session1*. So ideally the number of objects at this point in the .NET heap should be same as *Session1*. If not then the extra objects are the ones that are leaking, unless you expect them to be there for some reason , which as a programmer of your application must know (May be you are allocating, say, some global objects every time you create a new child window, and do not dispose them for some specific reason you know better. But all the other objects that you expect not to be there, but are there anyhow ,suggest that they are the ones actually leaking.). So proceed with the *same* set of functions as mentioned above in Pt.5 . Click **Capture**, **Save** (Save it as *Session2*) and **Clear**. Close the profiling control window and the *Scribble* application. So, Now we have 2 sessions to compare and find out how many (if at all ) and of what size , are the objects that remain in memory (i.e. leak) when you create a new Child MDI in the *Scribble* application.

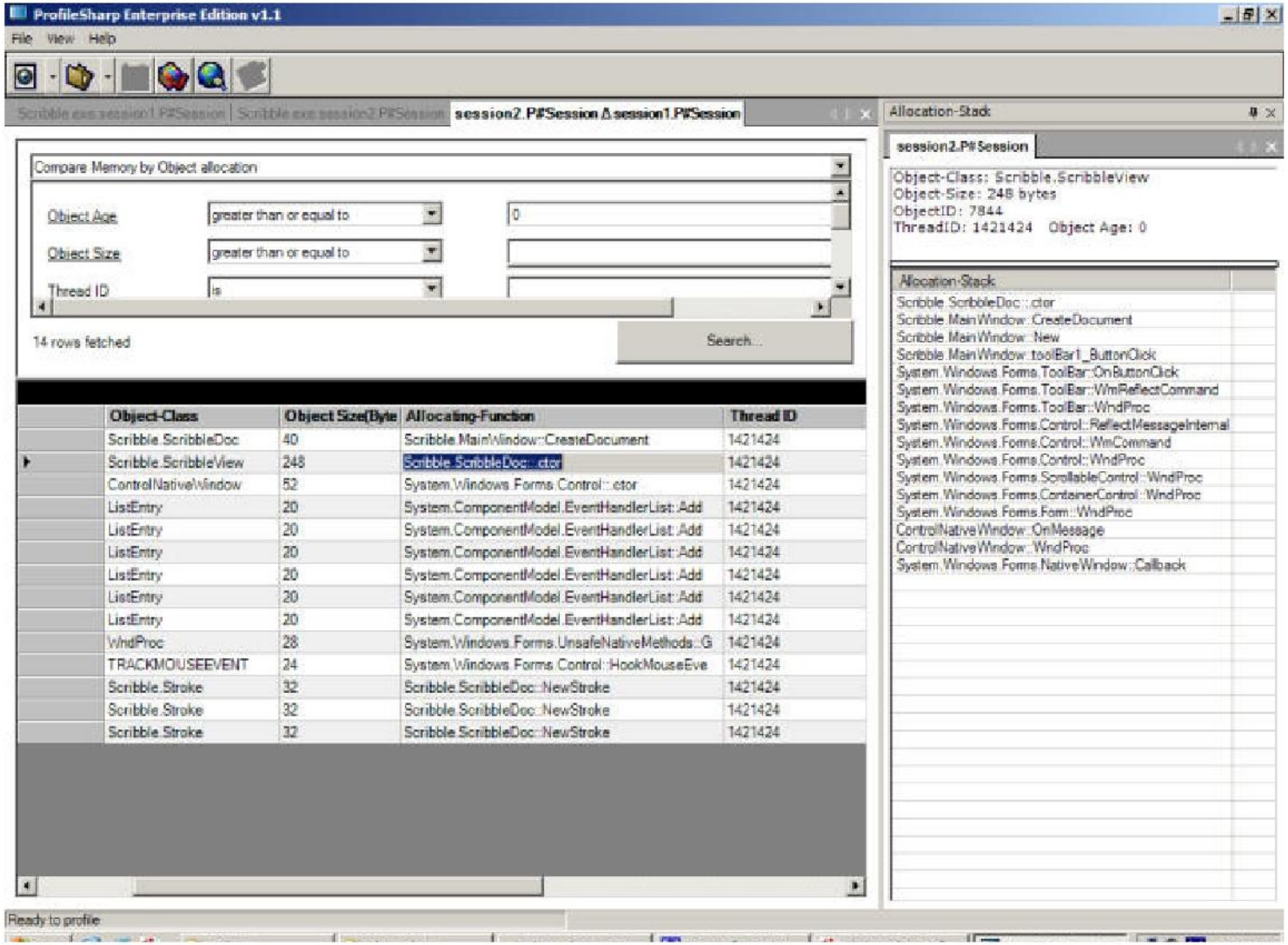
7.) Open the 2 sessions (*Session1* and *Session2*) one after the other in the profiler (**File->Open->Memory Analysis Session**..). Let the 2 sessions load. Select the menu '**File->Open->Memory Comparison Session**'. *Note that you must have at least 2 sessions already loaded (in this case Session1 and Session2) to actually start the* ***Comparison Session.***



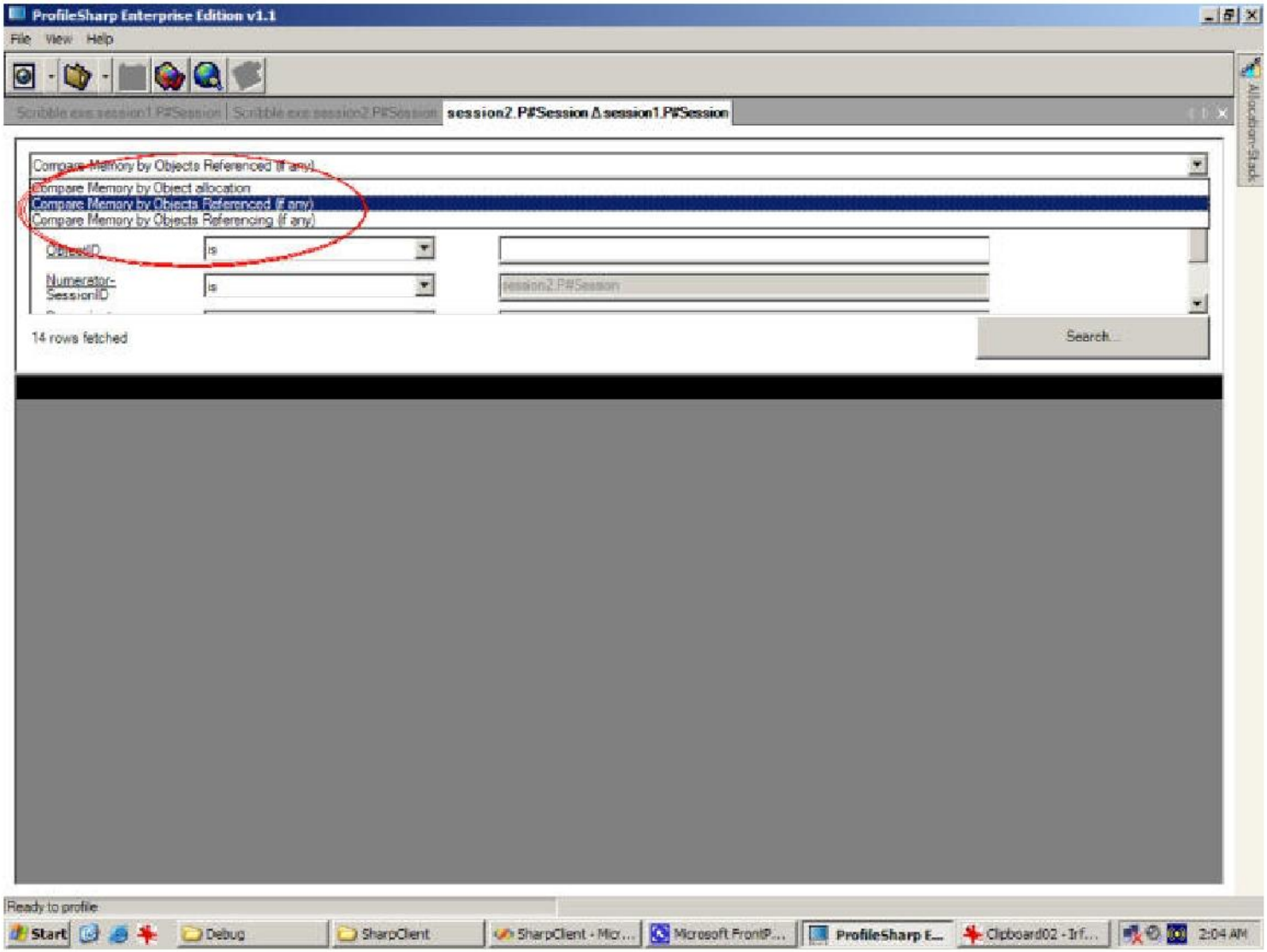


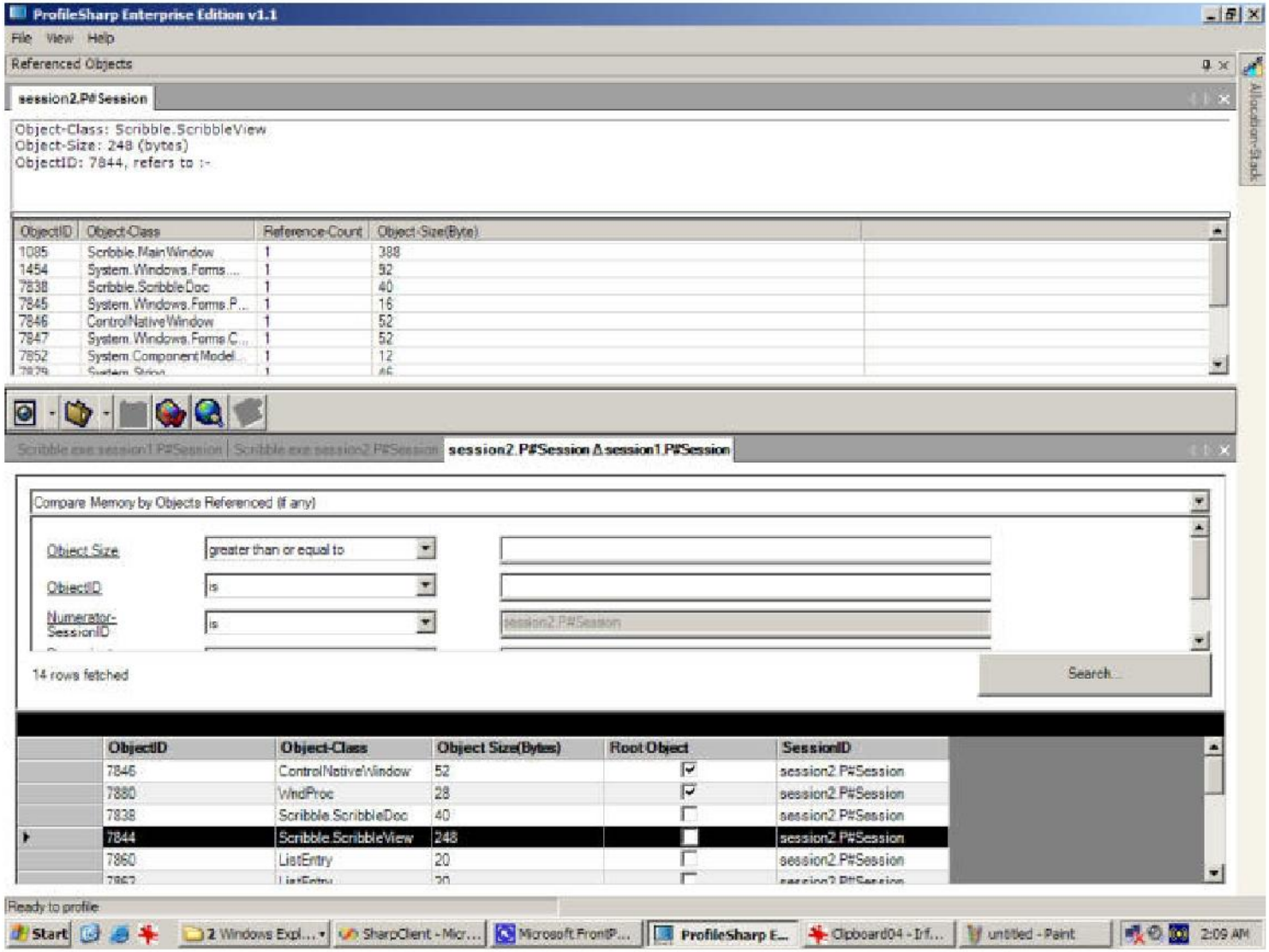

Here we choose *Session2* as our first option (the Numerator session) and *Session1* as option 2 (the Denominator session).

Click the **Compare...** button. The profiler now opens the report which you can use not only to get the information about the objects that can be considered as left-over (i.e. loitering or undisposed objects) by the numerator session (*Session2* in this case) over the denominator session (*Session1* in this case), but also the allocation-stack, referenced objects, referencing objects and all other pieces of information for all such leaking objects (Provided you had selected the corresponding option in the **Configurator** for each of the sessions before starting the profiling)



You can switch between the different reports/views for the comparison session by selecting the appropriate option in the drop-down combo-box above the Report-Grid.





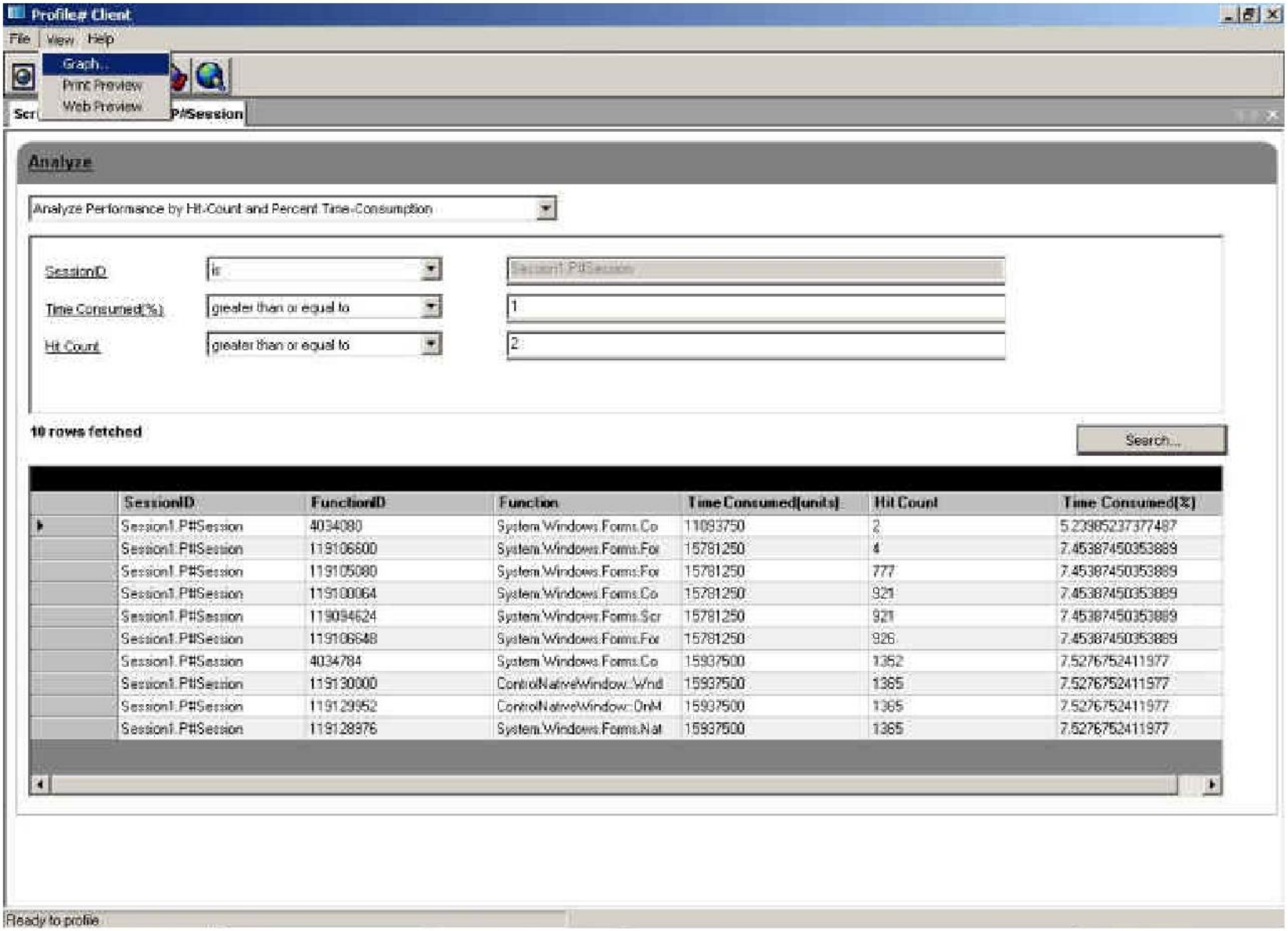

Apply the appropriate comparison view to your liking to get detailed information and to pinpoint the objects leaking between the 2 sessions.

*[Numerator Session :- A numerator session is one that is being compared i.e. the session that we want to analyze for objects that are in this session but not in the denominator session]
*[Denominator Session :- A denominator session is one that is being compared over i.e. the session with benchmarked number of objects]
Hence, note that swapping the numerator and the denominator session for a comparison session may lead to different set of results giving details of objects that were in *Session1* and not in *Session2*.

So here, we are done with the Memory Comparison Session as well.

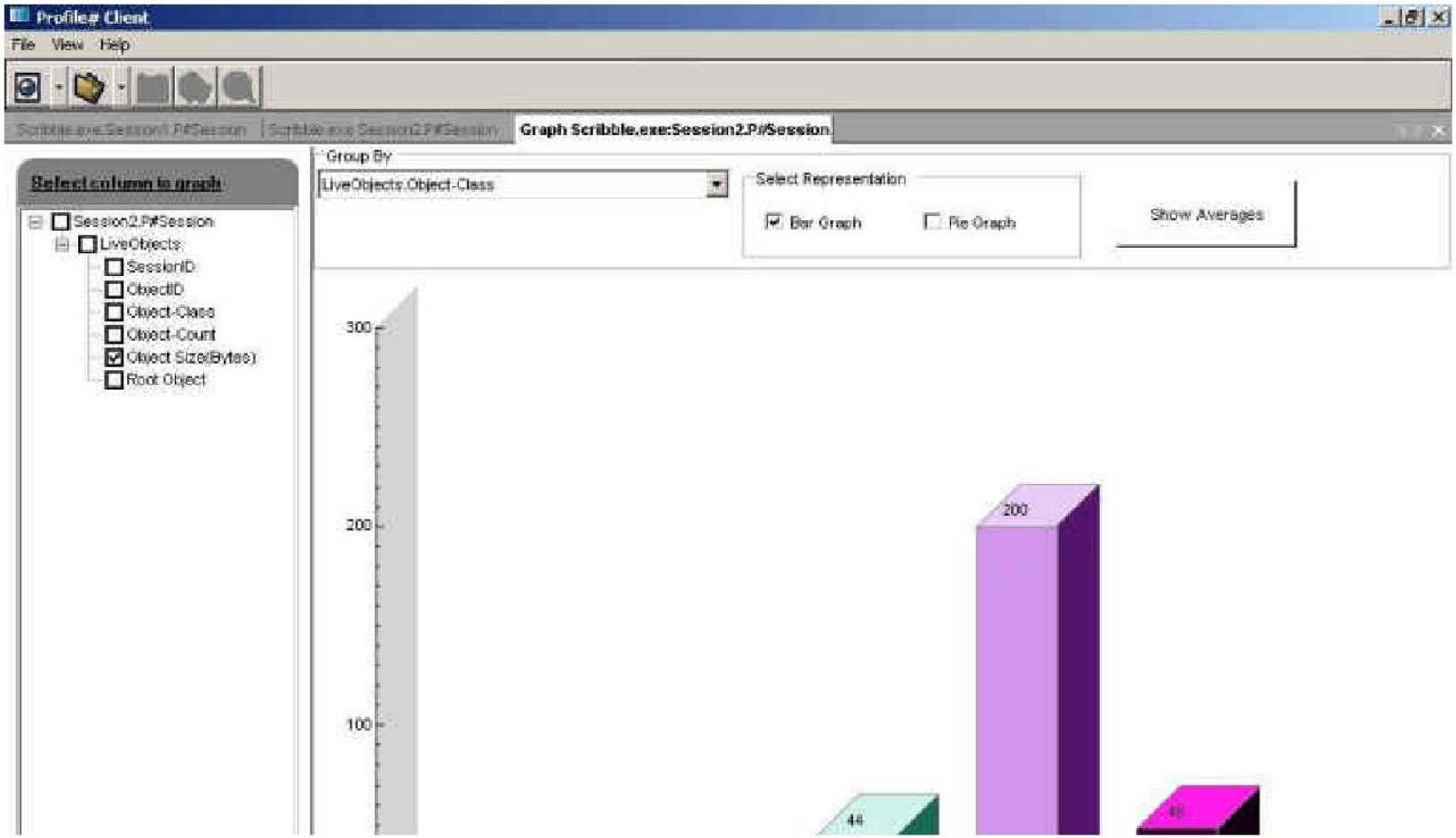
Making charts and graphs. [Steps to follow]

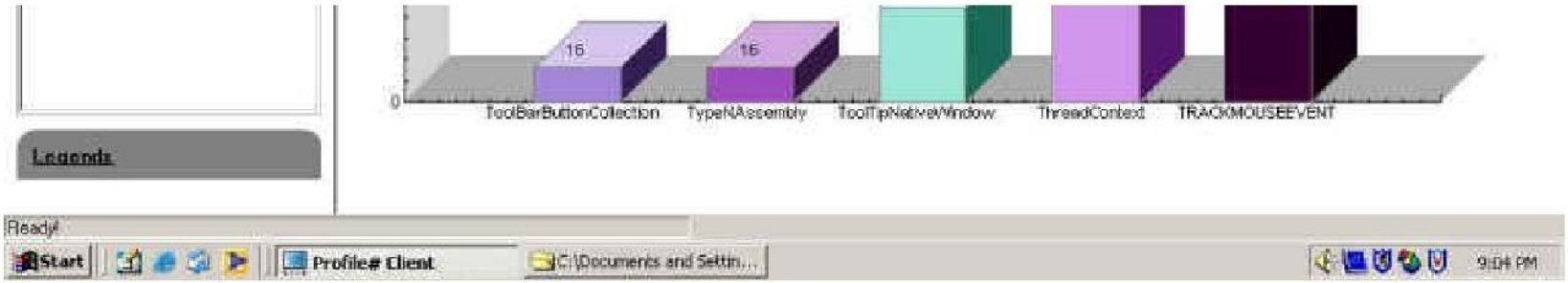
The act of creating Graphs and Web/Print Previews for the session's data being displayed in the **ReportGrid** is trivial. Whenever there is some data in the **ReportGrid** being displayed, the corresponding **Create Graph** and **Web/Print Preview** toolbar buttons as well as their menu buttons get enabled. Just press the appropriate button. This will take you to the window where you can choose what and how to Graph or Preview the session's data.



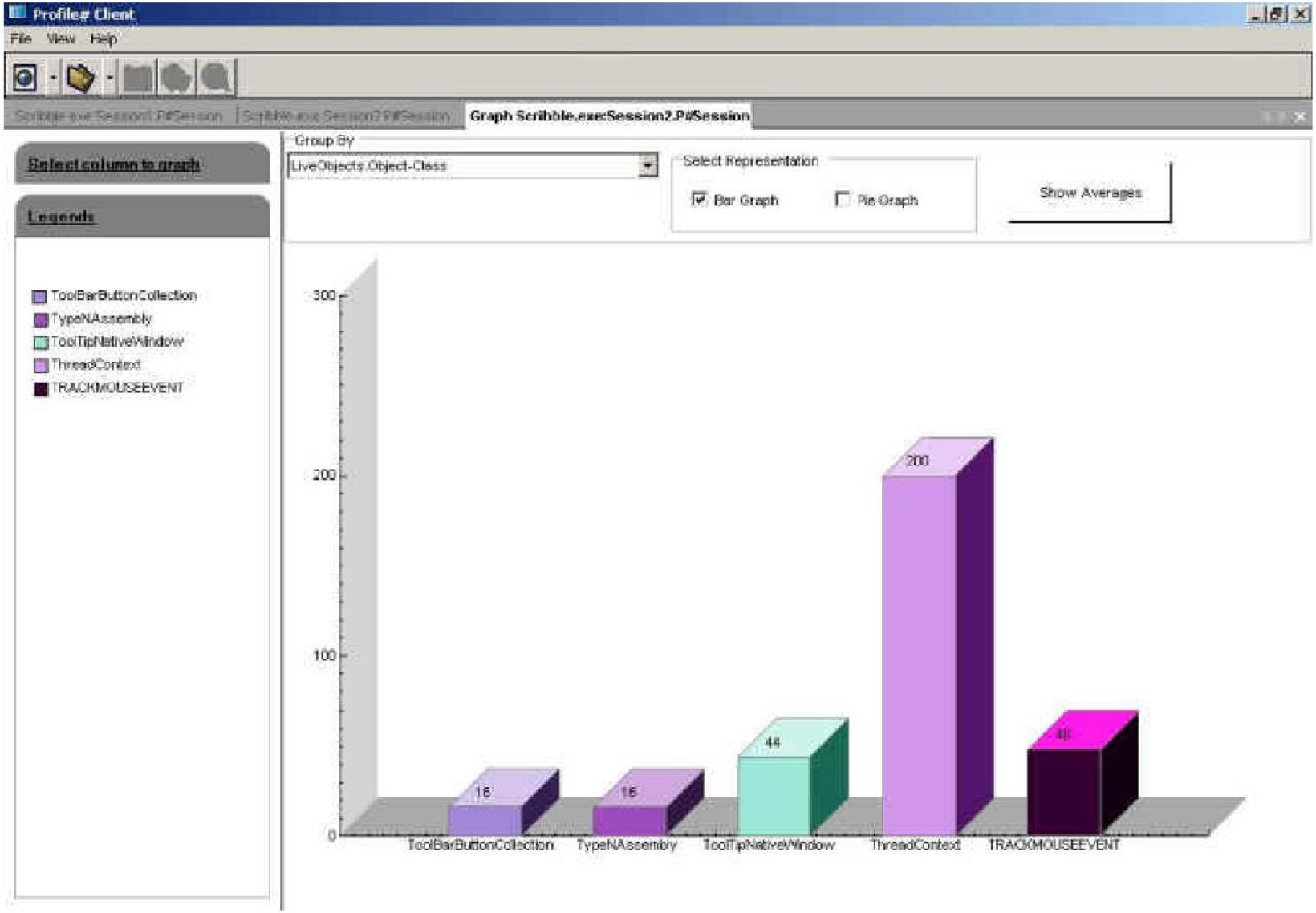

Let's create a graph first. Once you have some profiling data being displayed in the **ReportGrid**, press the **Create Graph** toolbar button or Select the menu '**View->Graph** '. In the resulting window, first of all select a column to graph from the tree-control in the sliding-tab window on the left. (as depicted below). The graph can only be created for a column that has only numeric values. So for now we select the column **Object-Size (Bytes)** (Obviously we are creating a graph for a Memory-Analysis Session in this case) .

This will populate the '**Group-By**' drop-down list box with all the columns available in the session's data. This list box is for selecting the column , by which the column-data that is to be graphed (**Object-Size (Bytes)** in this case), will be clubbed/grouped against . In most cases, this should be the column that represents the class-names (**Object-Class** for memory profiling session, and **Function-Name** when you are graphing a performance profiling session). Click the Bar-Graph or the Pie-Graph to view the graph. Just in case you are graphing a session that exceeds a limited number of rows, the graph created can be a bit cluttered due to excessive data. So in this case you are suggested that you get back to the Session's Data original tab and filter down the session's data to a smaller number of rows, by applying a filter criteria in the **ReportGrid**, and then retry with creating the graph with this trimmed down data.



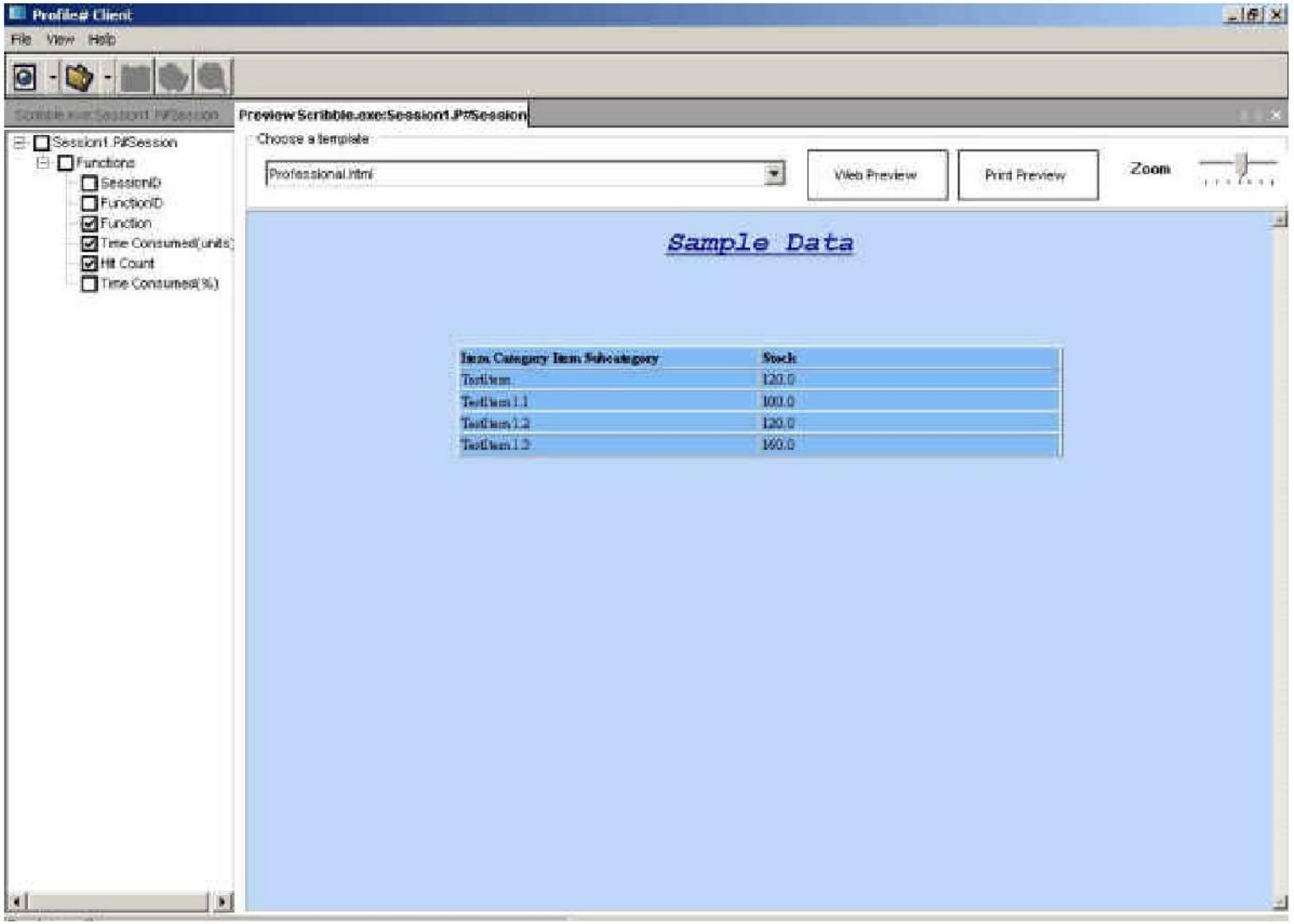



*You can click the **Legends** sliding-tab to see the color-mapping between the graph's X and Y axis.*



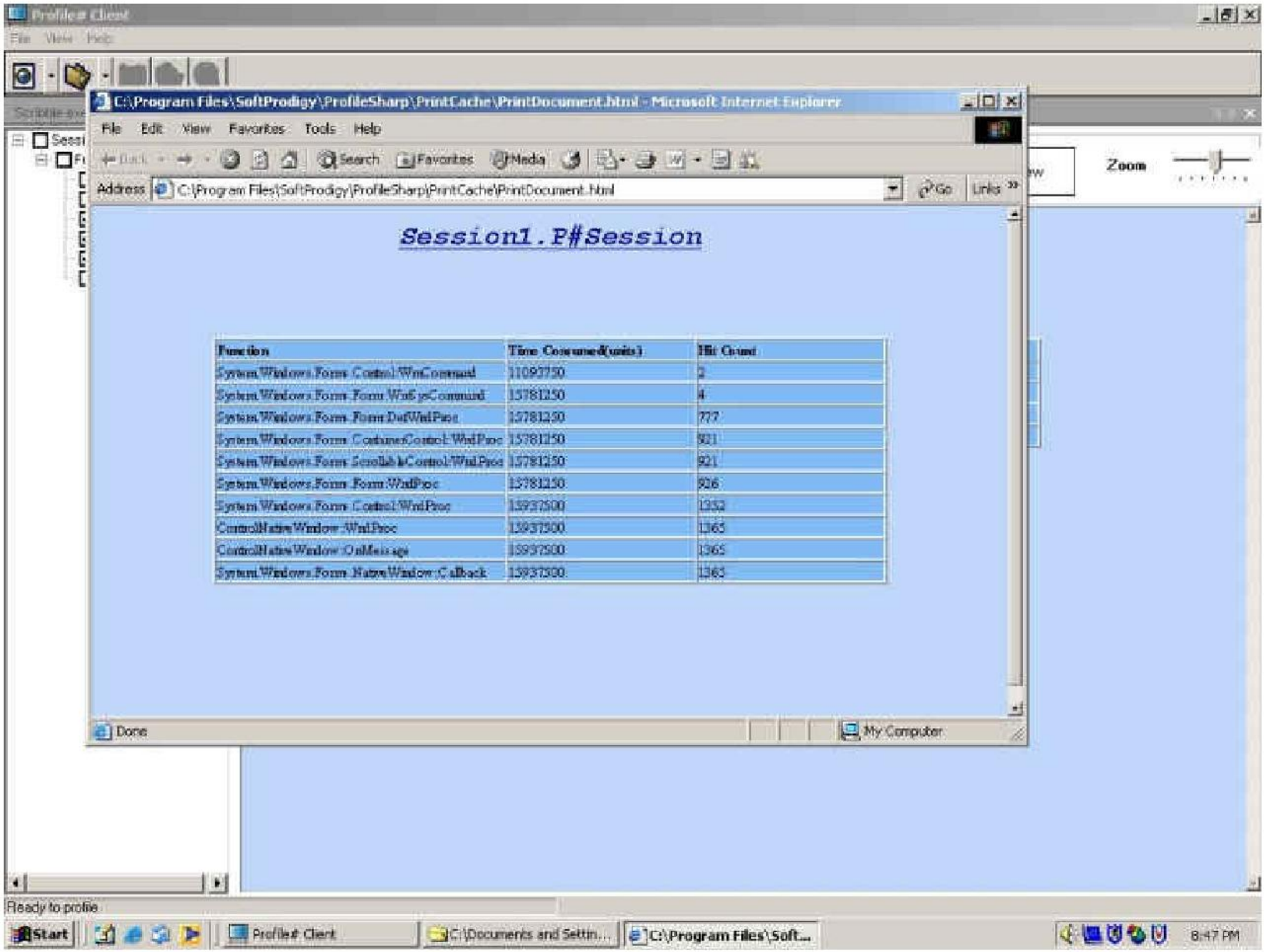

Creating a **Web/Print Preview** is even simpler. Just press the **Web/Print Preview** toolbar button when you have some preview-able data in the **ReportGrid** for a loaded session. In the resulting tab window, select all the columns that you want to **Web/Print-Preview** from the tree-view in the sliding-tab window on the left of your screen.

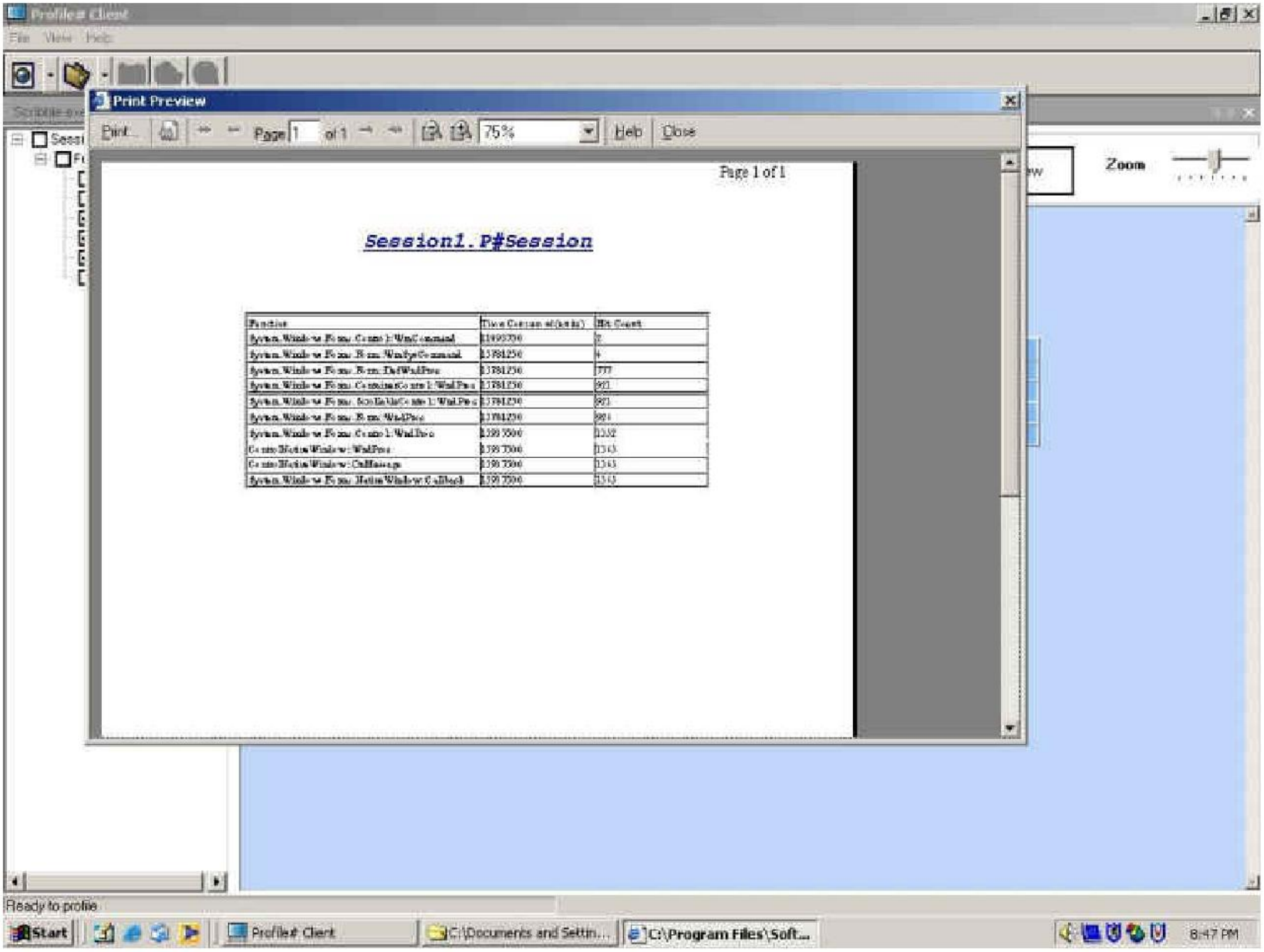
Choose a template from the dropdown list-box on the top of the Right aligned larger window. This will display some sample data , in a format specific to the template selected. Select the **Zoom** sliding control to adjust the size of data that shall be displayed, before previewing the session data.




Now click the **Web Preview** or the **Print Preview** button to apply the template to the session's data and form/display an HTML/Print report of it. You must have *Internet Explorer 6.0* or above to properly preview the data.

## Web Preview and Print Preview

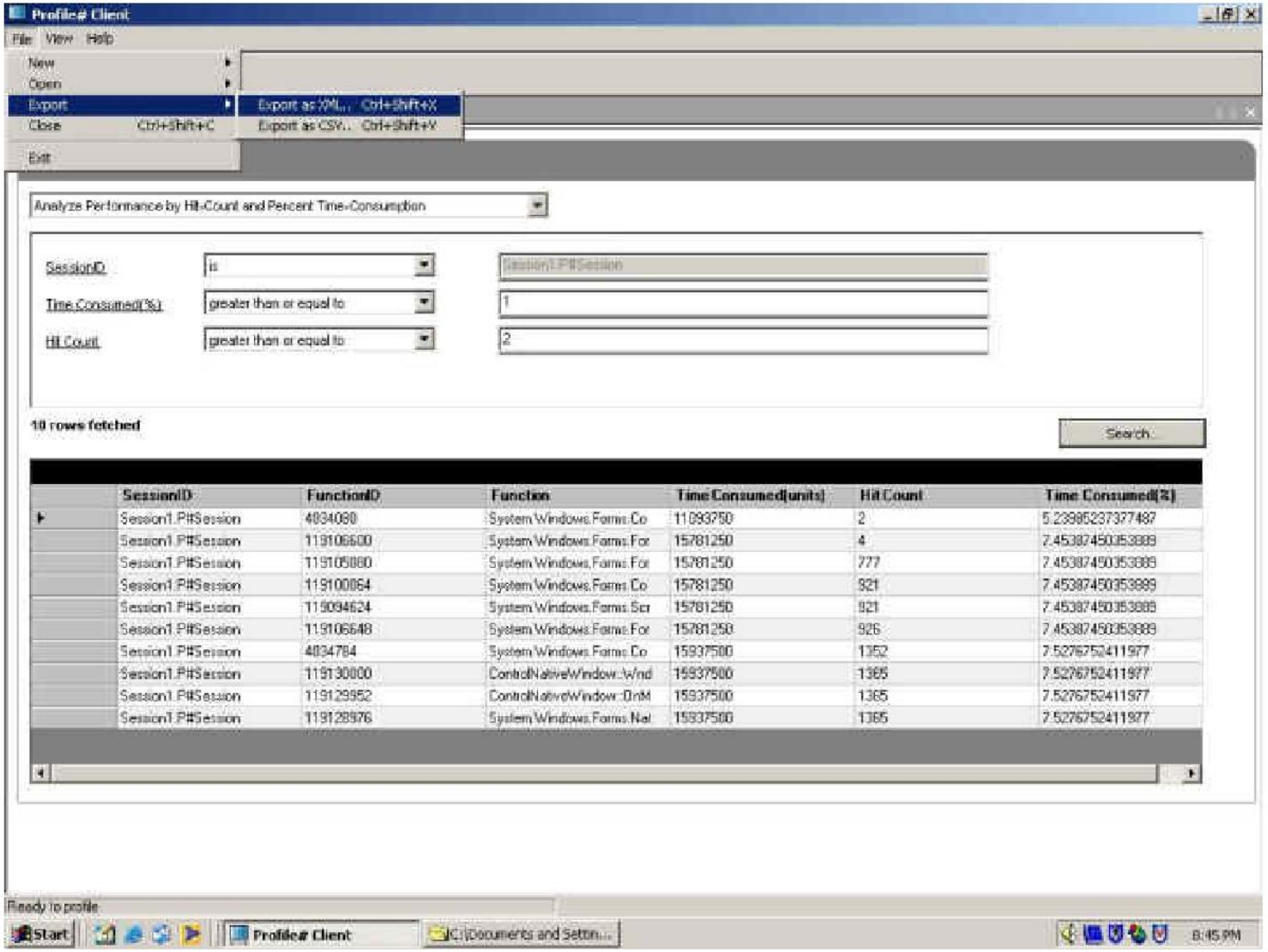





To export the data in XML or CSV (Comma Separated Values) format select the appropriate option from the menu 'File->Export->Export as XML..'
or

'File->Export->Export as CSV..', respectively. This option too will be automatically available when a session has been loaded for analysis and there

is some data available for display in the ReportGrid.




## FAQ

***ProfileSharp*** *Frequently Asked Questions*

**Q. What are the system requirements for me to use ProfileSharp?**

**A.**

- Windows 2000, Windows XP ,Windows 2003 Server
- ProfileSharp itself requires Microsoft .NET Framework version 1.1. It can however profile .NET applications built against version 2.0, 1.1 and 1.0.
- IE 6+ (Internet Explorer 6)
- Microsoft Jet Driver 4.0 (OdbcJt32.dll) [*Normally ships with the OS*]
- 128MB RAM, 5 MB hard disk space
- Runs on a common PC Workstation/machine.

**Q. Does ProfileSharp also profile pure COM or native (unmanaged) applications?**

**A.** No, ProfileSharp is a profiler for managed applications only. It however **can** profile unmanaged calls made or received by the CLR. [COM Interop or PInvoke] . But once the call enters inside the unmanaged domain, .NET profiler is not able to profile it . .NET components interacting with COM (COM Interop) or Native Win32 (PInvoke) can be profiled along with pure .NET assemblies and components. It can also profile unsafe methods written in .NET. (All in all , anything that compiles to MSIL is profilable)

**Q. What languages can I profile in with ProfileSharp?**

**A.** Any .NET compliant language [C#, VB.NET, VC++.NET (Managed C++), VJ# etc. ]

**Q. Why can't I see source code for lines profiled, when I profile an application?**

**A.** Check for following things :

- Is your Profilee application in the debug build?
- Are the debug symbol files (.PDB files) for all the assemblies you want to profile in the same directory as their executable counterparts (dlls, exe)

If you still can't see the source code, register 'mscordbi.dll' ,'diasymreader.dll' ,'mscorie.dll' ,mscormmc. dll' ,'mscorld.dll' and 'mscorsec.dll' as COM dlls (in the .NET framework's folder corresponding to the target runtime of your application ) [Say , if your application targets the v1.0.3705 version

(1.0) ,and C: \Winnt is your Windows directory then got to the C:\Winnt\Microsoft.NET\Framework\v1.0.3705 folder and register the mentioned dlls as normal COM dlls, if you want to revert back , just register the corresponding dlls in the original framework's folder. Also go to your Program Files directory and browse to the folder '**\Common Files\Microsoft Shared\VS7Debug'** . Run the executable '**vs7jit.exe**' from dos command-line with '**/RegServer**' as parameter Actually this needs to be done because Microsoft's .NET debugger when installed with a .NET version, un-registers the old version's corresponding dlls. ] We are addressing this issue and will be sorted out in the next version of the profiler.

However, if this too does not solve the problem, you may have to reinstall the corresponding .NET Frameworks.

**Q. Can I attach the profiler to an already running process?**

**A.** Practically, yes! [ **Other profilers do not give you that functionality** ].The only catch is , that you can attach the profiler to only those processes, which were started while the profiler was running [i.e. during the course of the time the profiler started, and till the profiler is closed.] All such processes will be listed down in the profiler to which you can attach the profiler. See technical documentation, accompanying.] However this functionality is only provided by the **Enterprise Edition** of **ProfileSharp.**

**Q. Can the profiler profile multithreaded and complex applications of huge size?**

**A.** Yes! All its versions can. Although we suggest the enterprise edition for heavy applications ,so that you can take the full advantage of all its features.

**Q. Why does the profiler slow down my application when it is profiling?**

**A.** The profiler inserts its own code which runs before and after every method is executed in your .NET application / service etc. so that it can get the necessary information that you desire from the profiler. Obviously this slows down your application. But the profiler takes into account the time that it takes itself and segregates and derives the time that your application actually would take in real time. So you always are presented with the real-time statistics for your application only.

**Q. I want to profile a .NET Windows Service from scratch. I use the "Suspend all new .NET processes.." option in the Profiler [Enterprise Edition] so that i can attach to the process and start profiling it right away. The problem is that the SCM [Service Control Manager] stops the service before I attach to it?**

A. The SCM only waits for 30 seconds for a service to start or stop. So you have to attach to the service within that time [Hey ! 30 seconds is enough] , if you want to profile a service right from the time CLR is first loaded into the service. Microsoft [as far as we know], does not support to elongate this time in the .NET Windows Services. The services built using native code [ATL etc.] have that option by virtue of a parameter **dwWaitHint.** So if you want to do so for your .NET services you will have to **PInvoke** all the necessary functionality into your .NET code to do it that way.

**Q. Why do I sometimes see garbage characters in the "Function-Name" and "Object-Class" column of the report-grid?**

A**.** If you have such problem, just do a simple remedy. Open the accompanying MS-Access DB [SharpBase.mdb]. Go to FTable table in design view and change the DataType of FSignature from Memo to Text [Make sure the Text size is 255,and it allows duplicate values]. This will solve the problem. Same can be applied to the 'ObjectName' field of 'LiveObjects' Table. Actually due to large signature of some of the profiled functions, this field has been set to memo by default so that it can contain function-signatures whose character length is above 255 [Limit value for 'Text' DataType in MS-Access],. This at random times shows irregular results. This issue will be fixed in the coming versions of ' **ProfileSharp** '.

**Q. When I open a new saved session for analysis, I sometimes see the analysis results irrelevant to the session opened?**

A. This is because you have most probably already loaded a previously saved profiler session into the profiler for analysis, which incidentally happens to have the **same name** as the new session you are trying to open (i.e. although the two sessions have different profiling data, they resemble in name.) Please note that you can and must only open those saved sessions at a time which have distinct names. The profiler will ignore and hence not load any new session for analysis if a session with the same name has already been loaded.

So **make sure all the profiling sessions you save, have distinct names** ,particularly the ones that you are going to open simultaneously in the profiler for analysis. The profiler caches the data at first load attempt on the basis of the session name and does not sport any other session with the same name till the previous synonymous session has been unloaded (i.e. till the profiler is restarted). However you can always reload the exactly same session more than once, in order to analyze it with a different perspective. [successive loading of the same session will be faster.]

***Q. When I start profiling an application, I get an error 'Unable to enumerate process modules. Unable to start profiling session...'?***

*A. Make sure you are logged in as an administrator. Also make sure that you have the following security settings enabled for the account that you are running the profiler from:-*

- Act as part of the operating system
- Create a token object.
- Debug Programs
- Profile Single Process
- Replace a process level token

[You can set these settings from the [Local Policies->User Rights Assignment] module of your System's [Control Panel->Administrative tools->Local Security Policy].

Alternatively type 'Secpol.msc' at command-line to access the 'Local Security Policy' control panel module.]

**Q. I am unable to save the profiling sessions?**

A. Make sure you have enough disk space. Make sure that the user account your **profilee** application [***not just the profiler***] is running under ,has sufficient rights to write to the folder that you want to save the profiler session to. Remember, in **ProfileSharp** the session data is saved by the **profilee** application and **not the profiler.**