Borland® Optimizeit™ Profiler

for the Microsoft® .NET Framework

Feature Matrix

Developer productivity

Flexibility to profile any .NET managed code
Improve performance and reliability of Microsoft® .NET managed code such as Visual Basic.NET, C#,® and Visual C++®
Borland® Optimizeit™ Profiler for Microsoft® .NET Framework fully supports the Common Language Runtime (CLR) for comprehensive performance profiling
Optimizeit Profiler supports Windows® 2000 (SP3), Windows XP™ (SP1), and the Microsoft .NET Framework 1.0 (SP1)

Ease of use
Plug and play: no need to recompile code or modify classes before execution
Start the tested application directly from the Optimizeit Profiler start panel
Program startup and configuration settings saved in setting files for quick startup of frequently tested programs
Option to start the tested program from the command line
Source code optional: Optimizeit Profiler provides performance and reliability information with no need for the source code. Source code access is useful to highlight relevant lines of code
Option to pause the tested program immediately after launch to study the program launch phase
Export profile data as ASCII, HTML, or easy-to-parse ASCII to print, compare, or archive data
Start/pause/stop the tested program at any time
Easily attach/detach to and from the application being profiled
Suspend/Resume CLR execution engine to start and stop the application
Visually isolate critical code with scalable call graphs. Select a string allocation and highlight the flow of a method call to see where memory and time are being spent
Extremely low overhead for analyzing the performance of applications under a simulated load and in a production environment.

Borland®

Comprehensive overview of performance hazards

High level Common Language Runtime (CLR) performance information
High-level performance overview allows developers to understand in real time if a performance issue is related to CPU, or memory, or both
Focus on a specific action in the application being tested by setting a reference mark and viewing the performance metrics from that point onwards
Class graph: displays the number of classes loaded in the Common Language Runtime (CLR)
Garbage Collector graph: tracks garbage collector activity
Thread graph: reveals the number of threads runnung and the number of threads actually using CPU

Memory profiling

Object allocation monitoring
Real-time monitoring of object allocations to understand in real time how the profiled program uses the CLR
Easy-to-read graph shows in real time all allocated instances per class with instance count and size
Graphical display clearly shows the volume of instances of each class
Option for filter class lists based on regular expressions to focus on relevant classes
Option to set a reference mark and view the instances allocated since the last mark, to focus on instances allocated by a specific action in the tested program
Option to view freed object count to understand performance issues related to the allocation of too many temporary objects
Garbage collection controls: Disable garbage collection to study temporary object allocations, enforce garbage collection at any time

Understand object allocations
Allocation Backtrace View allows developers to identify the code or part of the program responsible for instance allocations
Graph display of all methods involved in instance allocations of a given class with the volume of allocations in percentage
Option to reverse display to more easily focus on specific methods or lines of code responsible for object allocations
Listing of methods with highest allocation count to help identify single methods responsible for excessive allocation
Option to refine the display precision from methods to lines of code
Automatic highlight of the lines of code relevant to instance allocations with the Source Code Viewer.

CPU profiling

CPU profiling
Measure pure CPU usage or time usage during a profiling session
Display of profiling information per thread
Color highlight of threads that were busy during profiling session
Graph display of all significant methods or lines of code used during the profiling sessions
Option to refine the display precision from methods to lines of code
Time spent per method or line of code in percentage and absolute time
Hot Spot™ Display: lists methods where most time was spent to help identify bottlenecks due to single methods
Automatic highlight of relevant lines of code with the Source Code Viewer
Sampling-based profiler allows profiling of large amounts of code over a long period of time with millisecond precision while operating with low overhead
Option to control the granularity of sampling-based profiler output by tuning the sampling period