

Namespace Debugland

Classes

[Debugger](#)

Provides debugging utilities for tracking method execution, timing, SQL command execution, variable declaration, and control flow statements within a .NET application.

Class Debugger

Namespace: [Debugland](#)

Assembly: Debugland - Backup.dll








Provides debugging utilities for tracking method execution, timing, SQL command execution, variable declaration, and control flow statements within a .NET application.

```
public static class Debugger
```

Inheritance

[object](#)  ← Debugger

Inherited Members

[object.Equals\(object\)](#)  , [object.Equals\(object, object\)](#)  , [object.GetHashCode\(\)](#)  , [object.GetType\(\)](#)  , [object.MemberwiseClone\(\)](#)  , [object.ReferenceEquals\(object, object\)](#)  , [object.ToString\(\)](#) 

Remarks

The [Debugger](#) class assists developers in enhancing code visibility and identifying issues during development and testing phases by offering a range of debugging functionalities.

Key Features

- **Method Tracking** - Start and stop timers to monitor method execution time.
- **SQL Command Debugging** - : Monitor SQL command execution and connection status.
- **Variable Declaration** - Notify the declaration of variables, including names and values.
- **Control Flow Tracking** - Monitor the initiation and termination of control flow statements such as try-catch blocks, loops, and if statements.
- **Message Output** - Write custom messages to the debug window for informative purposes.

Usage of this class in a DEBUG environment enhances code visibility and aids in the identification and resolution of potential issues during development.

Methods

CatchBlockInitiated()

This Method is used to let you know that a Catch Block has been initiated.

```
[Conditional("DEBUG")]  
public static void CatchBlockInitiated()
```

CatchBlockTerminated()

This Method is used to let you know that a Catch Block has been terminated.

```
[Conditional("DEBUG")]  
public static void CatchBlockTerminated()
```

Close()

Flushes the output buffer and then calls the Close method on each of the Listeners. Basically the same as `debug.close();`

```
[Conditional("DEBUG")]  
public static void Close()
```

DoWhileLoopInitiated()

This Method is used to let you know that a Do-while loop has been initiated.

```
[Conditional("DEBUG")]  
public static void DoWhileLoopInitiated()
```

DoWhileLoopTerminated()

This Method is used to let you know that a Do-while loop has been terminated.

```
[Conditional("DEBUG")]  
public static void DoWhileLoopTerminated()
```

Fail(string, string)

Writes a failure message to the debug window.

```
[Conditional("DEBUG")]  
public static void Fail(string message, string secondMessage)
```

Parameters

message [string](#) 

The primary message to be written to the debug window.

secondMessage [string](#) 

An additional message to provide context or details about the failure.

FinallyBlockInitiated()

This Method is used to let you know that a Finally Block has been initiated.

```
[Conditional("DEBUG")]  
public static void FinallyBlockInitiated()
```

FinallyBlockTerminated()

This Method is used to let you know that a Finally Block has been terminated.

```
[Conditional("DEBUG")]  
public static void FinallyBlockTerminated()
```

ForLoopInitiated()

This Method is used to let you know that a for loop has been initiated.

```
[Conditional("DEBUG")]  
public static void ForLoopInitiated()
```

ForLoopTerminated()

This Method is used to let you know that a for loop has been terminated.

```
[Conditional("DEBUG")]  
public static void ForLoopTerminated()
```

IfInitiated()

This Method is used to let you know that a If Statement has been initiated.

```
[Conditional("DEBUG")]  
public static void IfInitiated()
```

IfTerminated()

This Method is used to let you know that a If Statement has been terminated.

```
[Conditional("DEBUG")]  
public static void IfTerminated()
```

Let(bool, string)

This method is used to check if a condition is true. If the condition is false, the method will write a message to the debug window.

```
[Conditional("DEBUG")]  
public static void Let(bool condition, string message)
```

Parameters

condition [bool](#)

The condition which is being checked.

message [string](#)

The message which is being written to the debug window.

Message(string)

Writes a message to the debug window.

```
[Conditional("DEBUG")]  
public static void Message(string message)
```

Parameters

message [string](#)

The message to be written to the debug window.

MessageIf(bool, object)

This method is used to write a message to the debug window if a condition is true.

```
[Conditional("DEBUG")]  
public static void MessageIf(bool condition, object value)
```

Parameters

condition [bool](#)

The conditional expression to evaluate. If the condition is true, the value is written to the trace listeners in the collection.

value [object](#)

An object whose name is sent to the Listeners.

MessageIf(bool, object, string)

This method is used to write a message to the debug window if a condition is true.

```
[Conditional("DEBUG")]  
public static void MessageIf(bool condition, object value, string category)
```

Parameters

condition [bool](#)

The condition which is being checked.

value [object](#)

An object whose name is sent to the Listeners.

category [string](#)

A category name used to organize the output.

MessageIf(bool, string)

This method is used to write a message to the debug window if a condition is true.

```
[Conditional("DEBUG")]  
public static void MessageIf(bool condition, string message)
```

Parameters

condition [bool](#)

The condition which is being checked.

message [string](#)

The message which is being written to the debug window.

MessageIf(bool, string, string)

This method is used to write a message to the debug window if a condition is true.

```
[Conditional("DEBUG")]
```

```
public static void MessageIf(bool condition, string stringMessage, string category)
```

Parameters

condition [bool](#)

The conditional expression to evaluate. If the condition is true, the category name and message are written to the trace listeners in the collection.

stringMessage [string](#)

A message to write.

category [string](#)

MessageImportant(string)

This method is used to write an important message to the debug window.

```
[Conditional("DEBUG")]  
public static void MessageImportant(string message)
```

Parameters

message [string](#)

The message which is being written to the debug window.

MethodInitiated(string)

Indicates that a method has started and initiated.

```
[Conditional("DEBUG")]  
public static void MethodInitiated(string methodName)
```

Parameters

methodName [string](#)

The name of the method being initiated.

Remarks

The method name is written to the debug window to provide context.

The debug indent level is adjusted to improve readability of debug output.

MethodTerminated(string)

Indicates the end of a method's execution.

```
[Conditional("DEBUG")]  
public static void MethodTerminated(string methodName)
```

Parameters

methodName [string](#)

The name of the method that has completed execution.

ReaderInitiated()

This Method shows that the SQL Reader has been initiated.

```
[Conditional("DEBUG")]  
public static void ReaderInitiated()
```

ReaderTerminated()

This method is used to debug SQL Commands, it will write the command to the debug window that the Reader has terminated.

```
[Conditional("DEBUG")]  
public static void ReaderTerminated()
```

SqlCommandInitiated(string)

This Method is used to debug SQL Commands, it will write the command to the debug window. This will initiate the SQL Connection.

```
[Conditional("DEBUG")]  
public static void SqlCommandInitiated(string operation)
```

Parameters

operation [string](#) 

The SQL Command which is being executed.

SqlCommandTerminated()

This method is used to debug SQL Commands, it will write the command to the debug window that the SQL Command has terminated.

```
[Conditional("DEBUG")]  
public static void SqlCommandTerminated()
```

SqlConnectionInitiated()

Connects to the SQL Server and the database. This method is used to debug SQL Commands, it will write the command to the debug window that the SQL Connection has been initiated.

```
[Conditional("DEBUG")]  
public static void SqlConnectionInitiated()
```

SqlConnectionTerminated()

This method is used to debug SQL Commands, it will write the command to the debug window that the SQL Connection has terminated.

```
[Conditional("DEBUG")]
```

```
public static void SqlConnectionTerminated()
```

TimeInitiated(string)

This method is used to start the stopwatch and write the elapsed time to the debug window.

```
[Conditional("DEBUG")]  
public static void TimeInitiated(string methodName)
```

Parameters

methodName [string](#)[↗]

The name of the method which is being called.

TimeTerminated(string)

This method is used to stop the stopwatch and write the elapsed time to the debug window.

```
[Conditional("DEBUG")]  
public static void TimeTerminated(string methodName)
```

Parameters

methodName [string](#)[↗]

The name of the method which is being called.

TryBlockInitiated()

This Method is used to let you know that a Try Block has been initiated.

```
[Conditional("DEBUG")]  
public static void TryBlockInitiated()
```

TryBlockTerminated()

This Method is used to let you know that a Try Block has been terminated.

```
[Conditional("DEBUG")]  
public static void TryBlockTerminated()
```

Variable()

This Method is used to let you know that multiple Variables has been declared.

```
[Conditional("DEBUG")]  
public static void Variable()
```

Variable(string)

This Method is used to let you know that a Variable has been declared.

```
[Conditional("DEBUG")]  
public static void Variable(string variableName)
```

Parameters

variableName [string](#) 

Name of the variable you declared

Variable(string, string)

This Method is used to let you know that a Variable has been declared. It also writes the value of the variable.

```
[Conditional("DEBUG")]  
public static void Variable(string variableName, string variableValue)
```

Parameters

`variableName` [string](#)

Name of the variable you declared

`variableValue` [string](#)

Value of the variable you declared

WhileLoopInitiated()

This Method is used to let you know that a while loop has been initiated.

```
[Conditional("DEBUG")]  
public static void WhileLoopInitiated()
```

WhileLoopTerminated()

This Method is used to let you know that a while loop has been terminated.

```
[Conditional("DEBUG")]  
public static void WhileLoopTerminated()
```