Namespace Debugland

Classes

Debugger

This class is used for debugging purposes.

Class Debugger

Namespace: Debugland

Assembly: Debugland - Backup.dll

This class is used for debugging purposes.

```
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()

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)

This method is used to write a fail message to the debug window.

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

Parameters

`message` string

The message which is being written to the debug window.

`secondMessage` string

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)

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

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

Parameters

`message` string

Beskeden som skal skrives til debug vinduet.

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)

This method indicates that the method has started and initiated. Begins the Stopwatch object and writes the name of the method to the debug window.

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

Parameters

`Name` string

MethodTerminated(string)

This method indicates that the method has ended. Also stops the Stopwatch object and writes the lifespan of the method to the debug window.

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

Parameters

`methodName` string

The name of the method which is being called.

ReaderInitiated()

This Method shows that the SQL Reader has been initiated.

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

```
public static void ReaderInitiated()
```

ReaderTerminating()

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 ReaderTerminating()
```

SQLCommandInitiated(string)

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

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

Parameters

`operation` string

The SQL Command which is being executed.

SQLCommandTerminating()

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 SQLCommandTerminating()
```

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()
```

SQLConnectionTerminating()

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 SQLConnectionTerminating()
```

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()
```