TimerMacro

The timer macro times how long it takes to execute a block of code.

See the Boo benchmarks page for examples using this macro.

The original JIRA issue has the C# source for this macro. It has been converted to the boo code below, but has not been tested yet (also the name was changed):

```
#region license
// Copyright (c) 2004, William P Wood
// All rights reserved.
//
// Redistribution and use in source and
binary forms, with or without modification,
// are permitted provided that the following
conditions are met:
//
//     * Redistributions of source code must
retain the above copyright notice,
//     this list of conditions and the
following disclaimer.
//     * Redistributions in binary form must
reproduce the above copyright notice,
//     this list of conditions and the
following disclaimer in the documentation
//     and/or other materials provided with
the distribution.
//     * Neither the name of Rodrigo B. de
Oliveira nor the names of its
//     contributors may be used to endorse
or promote products derived from this
//     software without specific prior
written permission.
```

```
//
// THIS SOFTWARE IS PROVIDED BY THE
COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND
// ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
// WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE
// DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
// FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
// DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR
// SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER
// CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY,
// OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF
// THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
#endregion

namespace Boo.Lang.Utils.Macros

import System
import Boo.Lang.Compiler
import Boo.Lang.Compiler.Ast
```

```
class TimeBlockMacro(AbstractAstMacro):
 override def Expand(macro as
MacroStatement) as Statement:
  argc as int = macro.Arguments.Count
  argIndex as int = 0
  localIndex as int = _context.AllocIndex()
  stmt as Block = Block()
  macroBlock as Block = macro.Block
  msg as Expression =
StringLiteralExpression('Elapsed time: ')
  userMsg as Expression = null
  elapsed as Expression =
ReferenceExpression(macro.LexicalInfo,
string.Format('__elapsedTime{0}__',
localIndex))
  start as Expression =
ReferenceExpression(macro.LexicalInfo,
string.Format('__startTime{0}__',
localIndex))
  dateNow as Expression =
AstUtil.CreateReferenceExpression('date.Now'
)
  argError as string = 'format is \'timer
[nLoops [, unroll]] [, "Message: "] [,
elapsedTimeVar]\''
  UNROLL as int = 8
  if argc > argIndex and
macro.Arguments[argIndex] isa
IntegerLiteralExpression:
   nLoops as IntegerLiteralExpression =
macro.Arguments[Math.Min(argIndex,
```

```
++argIndex)]
   msg =
StringLiteralExpression(string.Format('Elaps
ed time for {0} executions: ',
nLoops.Value))
   if argc > argIndex and
macro.Arguments[argIndex] isa
IntegerLiteralExpression:
    macroBlock = CreateTimerBlock(macro,
nLoops, cast(IntegerLiteralExpression,
macro.Arguments[Math.Min(argIndex,
++argIndex)]), localIndex)
   else:
    macroBlock = CreateTimerBlock(macro,
nLoops, IntegerLiteralExpression(UNROLL),
localIndex)


  if argc > argIndex and
macro.Arguments[argIndex] isa
StringLiteralExpression:
   userMsg =
macro.Arguments[Math.Min(argIndex,
++argIndex)]

  if argc > argIndex and
macro.Arguments[argIndex] isa
ReferenceExpression:
   elapsed =
macro.Arguments[Math.Min(argIndex,
++argIndex)]
```

```
    msg = null

   if argc != argIndex:
    raise System.ArgumentException(argError)

   if userMsg != null:
    msg = userMsg


stmt.Add(BinaryExpression(macro.LexicalInfo,
BinaryOperatorType.Assign, start, dateNow))
  stmt.Add(macroBlock)
  elapsedCalc as Expression =
BinaryExpression(macro.LexicalInfo,
BinaryOperatorType.Subtraction,
dateNow.CloneNode(), start.CloneNode())

stmt.Add(BinaryExpression(macro.LexicalInfo,
BinaryOperatorType.Assign, elapsed,
elapsedCalc))
  if msg != null:

stmt.Add(AstUtil.CreateMethodInvocationExpre
ssion(macro.LexicalInfo,
AstUtil.CreateReferenceExpression('System.Co
nsole.Write'), msg))

stmt.Add(AstUtil.CreateMethodInvocationExpre
ssion(macro.LexicalInfo,
AstUtil.CreateReferenceExpression('System.Co
nsole.WriteLine'), elapsed))
```

```
  return stmt

 private def CreateTimerBlock(macro as
MacroStatement, nLoops as
IntegerLiteralExpression, unroll as
IntegerLiteralExpression, localIndex as int)
as Block:
  stmt as Block = Block(macro.LexicalInfo)
  whileStmt as WhileStatement =
WhileStatement(macro.LexicalInfo)
  loopCounter as ReferenceExpression =
ReferenceExpression(string.Format('__loopCou
nter{0}__', localIndex))
  nLoopsVal as long = nLoops.Value
  unrollVal as long = unroll.Value
  unrollVal = 1 if unrollVal < 1
  if nLoopsVal >= 2 * unrollVal:

stmt.Add(BinaryExpression(macro.LexicalInfo,
BinaryOperatorType.Assign, loopCounter,
IntegerLiteralExpression(0)))
   nLoops.Value = nLoops.Value /
unroll.Value
   whileStmt.Condition =
BinaryExpression(macro.LexicalInfo,
BinaryOperatorType.LessThan,
loopCounter.CloneNode(), nLoops)

whileStmt.Block.Add(UnaryExpression(UnaryOpe
ratorType.Increment,
```

```
loopCounter.CloneNode()))
   i as int = 0
   while i < unrollVal:

whileStmt.Block.Add(macro.Block.CloneNode())
    ++i

   nLoopsVal -= nLoops.Value * unroll.Value
   stmt.Add(whileStmt)

  while nLoopsVal > 0:
   stmt.Add(macro.Block.CloneNode())
   --nLoopsVal
```

```
return stmt
```

See also the --profile option for the mono command line tool to profile your application for bottlenecks.