fireAndForget

The fireAndForget macro automatically calls the appropriate EndMethod of an asychronous call.

Example:

```
class Foo:
  [AsyncMethod]
  def PrintMessage(message):
    print message

def Main():
  foo = Foo()

  fireAndForget foo.BeginPrintMessage("Hello
World!")
```

```
#region license
// Copyright (c) 2005, Sorin Ionescu
// 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 Sorin Ionescu
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.
#endregionTestFixtureTestFixtureTestFixtureT
estFixture

namespace Boo.Lang.Useful.Macros

import Boo.Lang.Compiler
import Boo.Lang.Compiler.Ast

class FireAndForgetMacro(AbstractAstMacro):
"""
Automatically calls the End<Method> of an
asynchronous invocation in a closure.
"""
 public override def Expand(macro as
MacroStatement) as Statement:
  assert macro.Arguments.Count ==  1
  assert macro.Arguments[0] isa
MethodInvocationExpression

  lexicalInfo = macro.LexicalInfo
  statement = Block(lexicalInfo)
  localIndex = Context.AllocIndex()
  beginMethodInvocation = macro.Arguments[0]
as MethodInvocationExpression
  beginMethodTarget = \
   beginMethodInvocation.Target as
MemberReferenceExpression
```

```
  objectReference = beginMethodTarget.Target
  endMethodName = "End" +
beginMethodTarget.Name.Substring(5)
  endMethodInvocation =
MethodInvocationExpression(
   LexicalInfo: lexicalInfo,
   Target: MemberReferenceExpression(
    objectReference.CloneNode(),
    endMethodName))

  callbackClosure =
CallableBlockExpression(lexicalInfo)
  iAsyncResult = DeclarationStatement(
   LexicalInfo: lexicalInfo,
   Declaration(LexicalInfo: lexicalInfo,
    Name: "___iAsyncResult${localIndex}",
    Type: SimpleTypeReference(lexicalInfo,
"System.IAsyncResult")),
   NullLiteralExpression(lexicalInfo))

  iAsyncResultAssignment = BinaryExpression(
   lexicalInfo,
   BinaryOperatorType.Assign,
   ReferenceExpression(
    LexicalInfo: lexicalInfo,
    Name: iAsyncResult.Declaration.Name),
   beginMethodInvocation)


beginMethodInvocation.Arguments.Add(callback
Closure)
```

```
beginMethodInvocation.Arguments.Add(NullLite
ralExpression(lexicalInfo))

  endMethodInvocation.Arguments.Add(
   ReferenceExpression(
    LexicalInfo: lexicalInfo,
    Name: iAsyncResult.Declaration.Name))


callbackClosure.Body.Add(endMethodInvocation
)

  statement.Add(iAsyncResult)
  statement.Add(iAsyncResultAssignment)
```

return statement