A Wrapper Generator

U. Kastens

April 22, 2010

1 Introduction

This text contains a specification for a Wrapper Generator to be created using the Eli system. The Wrapper Generator can be used to generate wrapper classes for given types.

The input for the generator is a sequence of type names. A C++ module is generated containing a wrapper class for each type name. Each object of such a class wraps a value of the type the class is created for. A common base class, named Object, is created for all those wrapper classes.

This specification is written in form of a FunnelWeb file. The documentation of FunnelWeb can be found under Eli/Tools/FunnelWeb. FunnelWeb can create both from this file, a specification used by Eli to create the Wrapper Generator and a documentation that can be processed by LaTex. This technique is called +Literate Programming+.

This file contains specification fragments, written as FunnelWeb macros, and plain text explaining them. The documentation can be created using LaTex. For that pupose the input for LaTex is derived by the Eli command: `Wrapper.fw:fwTex`; the command `Wrapper.fw:fwTex:pdf` is used to create a pdf file for it.

The generator specification is extracted automatically by Eli from this file when Eli is asked to derive a product from it, e.g. the executable processor using the command `Wrapper.fw:exe`.

The following fragment is an example for an input to the Wrapper Generator. It is a FunnelWeb macro which creates a file named `example` (see FunnelWeb/Output Files).

example[1]:

```
int;
PairPtr; "Pair.h"
```

This macro is attached to a product file.

The following fragment creates a file named `Wrapper.con`; it is a specification of type `con`. Specifications of that type are used to describe the concrete syntax of the generator's input. In the Eli Tutorial

`Guide for New Eli Users/Descriptive Mechanisms Known to Eli` those types are explained briefly, and references to documents are given which give detailed descriptions how such specifications are used.

In this case specifications of concrete syntax are explained in the document `Syntactic Analysis`.

Wrapper.con[2]:

```
Specification:  Sequence.
Sequence:       Sequence Element / .
Element:        TypeName ';'.
Element:        FileName.
TypeName:       Identifier.
FileName:       String.
```

This macro is attached to a product file.

Wrapper.gla[3]:

```
Identifier: C_IDENTIFIER
String: C_STRING_LIT
C_COMMENT
```

This macro is attached to a product file.

FileName[4]:

```
SYMBOL FileName: Include: PTGNode;
SYMBOL FileName COMPUTE
  SYNT.Include = PTGInclude (StringTable (TERM));
END;
```

This macro is invoked in definition 11.

TypeName[5]:

```
SYMBOL TypeName:
         KindDef, ClassFwd, ObjectGet, ClassHead, ClassGet: PTGNode;

SYMBOL TypeName COMPUTE
  SYNT.KindDef =   PTGKindDef (StringTable (TERM), THIS.KindNumber);
  SYNT.ClassFwd = PTGClassFwd (StringTable (TERM));
  SYNT.ObjectGet = PTGObjectGet (StringTable (TERM));
  SYNT.ClassHead = PTGClassHead (StringTable (TERM));
  SYNT.ClassGet =  PTGClassGet (StringTable (TERM));
END;
```

This macro is invoked in definition 11.

KindNumbers[6]:

```
CHAIN KindNumber: int;

SYMBOL Specification COMPUTE
  CHAINSTART HEAD.KindNumber = 1;
END;

SYMBOL TypeName COMPUTE
  THIS.KindNumber = ADD (THIS.KindNumber, 1);
END;
```

This macro is invoked in definition 11.

Wrapper[7]:

```
RULE: Specification LISTOF Element COMPUTE
   PTGOutFile
     (CatStrStr (SRCFILE, ".h"),
      PTGWrapperHdr
        (CONSTITUENTS FileName.Include
           WITH (PTGNode, PTGSeq, IDENTICAL, PTGNull),
         CONSTITUENTS TypeName.KindDef
           WITH (PTGNode, PTGSeq, IDENTICAL, PTGNull),
         CONSTITUENTS TypeName.ClassFwd
           WITH (PTGNode, PTGSeq, IDENTICAL, PTGNull),
         CONSTITUENTS TypeName.ObjectGet
           WITH (PTGNode, PTGSeq, IDENTICAL, PTGNull),
         CONSTITUENTS TypeName.ClassHead
           WITH (PTGNode, PTGSeq, IDENTICAL, PTGNull)));

   PTGOutFile
     (CatStrStr (SRCFILE, ".cc"),
      PTGWrapperImpl
        (SRCFILE,
         CONSTITUENTS TypeName.ClassGet
           WITH (PTGNode, PTGSeq, IDENTICAL, PTGNull)));
END;
```

This macro is invoked in definition 11.

Wrapper.specs[8]:

```
$/Tech/Strings.specs
$/Output/PtgCommon.fw
```

This macro is attached to a product file.

Wrapper.HEAD.phi[9]:

```
#include "source.h"
```

This macro is attached to a product file.

Wrapper.ptg[10]:

```
Include:
  "#include " $ string "\n"

KindDef:
  "#define " $ string "Kind \t" $ int "\n"

ClassFwd:
  "class " $ string "Wrapper;\n"

ObjectGet:
  "  " $1 string " get" $1 string "Value ();\n"

ClassHead:
  "class " $1 string "Wrapper : public Object {\n"
  "private:\n"
  "    " $1 string " v;\n"
  "public:\n"
  "    " $1 string "Wrapper (" $1 string " value) "
       "{ kind = " $1 string "Kind; v = value; }\n"
  "    " $1 string " getValue () { return v; }\n"
  "};\n\n"

ClassGet:
  $1 string " Object::get" $1 string "Value () {\n"
  "      if (kind == " $1 string "Kind)\n"
  "         return ((" $1 string "Wrapper*)this)->getValue();\n"
  "      else\n"
  "         throw WrapperExcept();\n"
  "}\n\n"

WrapperHdr:
  "#ifndef WRAPPER_H\n"
```

```
  "#define WRAPPER_H\n\n"
  $1 /* Includes */
  "\n#define noKind      0\n"
  $2 /* KindDefs */
  "\n"
  $3 /* ClassFwds */
  "\n"
  "class Object {\n"
  "public:\n"
  "  class WrapperExcept {};\n"
  "  int getKind () { return kind; }\n"
  $4 /* ObjectGets */
  "protected:\n"
  "  int kind;\n"
  "};\n\n"
  $5 /* ClassHeads */
  "\n#endif\n"

WrapperImpl:
  "#include \"" $ string ".h\"\n\n"
  $ /* ClassGets */
```

This macro is attached to a product file.

Wrapper.lido[11]:

Wrapper[7]
FileName[4]
TypeName[5]
KindNumbers[6]

This macro is attached to a product file.