

cegen—a freeware C++ generator for Rational Rose *

Claude Eisenhut
ceisenhut@eisenhutinformatik.ch

Version 1.2.0
February 4, 2000

Abstract

This document describes cegen. It covers installation, usage, code generation properties, generated code and source structure.

1 Introduction

This is a C++ code generator for Rational Rose. Key features are

- Sourcecode is available.
- Code regeneration is possible.
- Parameters are passed by reference.
- Association manipulators are generated.

This code generator now does most things; however, there are some missing features among

- templates
- enumerations
- nested classes
- association classes
- multi attribute qualifiers

*Copyright (c) 1999-2000, Claude Eisenhut All rights reserved. See the file license.txt for the specific language governing rights and limitations under the license.

1.1 Website/Mailing lists

Visit the page at

<http://www.eisenhutinformatik.ch/rose/freeware/default.htm>

for information about downloads, new releases, patches and so on of this and other packages.

To learn of new scripts and versions please subscribe to the `ceroses` mailing list. To subscribe send an e-mail with the body

```
subscribe ceroses
```

To unsubscribe send an e-mail with the body:

```
unsubscribe ceroses
```

to `ceroses-request@eisenhutinformatik.ch`

To discuss the use of `cegen` please subscribe to the `cegen` mailing list. To subscribe send an e-mail with the body

```
subscribe cegen
```

To unsubscribe send an e-mail with the body:

```
unsubscribe cegen
```

to `cegen-request@eisenhutinformatik.ch`

2 Installation

Copy the files `cegen.ebs`, `cegen.ptx`, `cegen.mnu` and `cegen.reg` from subdirectory `src` to a new directory (e.g. `d:\rose98i\cegen\`).

Edit the menu file. In file `d:\rose98i\cegen\cegen.mnu` you should change the following line.

```
RoseScript d:\rose98i\cegen\cegen
```

Update the registry. In file `d:\rose98i\cegen\cegen.reg` you should change the following line. After you saved the file, double click it or import it with `regedit`.

```
"InstallDir"="d:\\rose98i\\cegen"
```

3 Usage

Select modules in a component diagram and run the script. The code generator will create backup files in the `bak` subdirectory of the generated modules.

4 Guide to the generated code

This code generator only translates the structural part of a model to C++ code. In order to get a running program you need to modify the generated code, at least add instructions to your defined operations. This section explains where and how to edit the generated code.

Some lines of the generated code are enclosed by marked C++ comment lines:

```
// -beg- preserve=no 37EB601501A6 codebody "A::ops"  
// -end- 37EB601501A6 codebody "A::ops"
```

These marked C++ comment lines function as tags to help the code generator match code sections with model elements. `37EB601501A6` is the unique identifier of the corresponding rose model element, `codebody` is the code generator name of the section, `"A::ops"` is the user readable name of the rose model element. `preserve` is used to indicate whether the contents of the section are to be preserved when code is regenerated. If you set it to `yes` the code generator will keep the contents of the section when source code is regenerated.

When editing generated code files:

- Edit only between the `// -beg-` and `// -end-` marked lines. Any code changes outside of these tags are not preserved.
- Set `preserve=yes`. Any other value of this argument will not preserve the section contents.
- Do not create your own sections — you can only use those provided by the code generator.
- Do not move code sections.
- Do not delete sections. But you may delete the contents of a section and set `preserve=yes`.

There are sections which you are not expected to change. If you change them you may lose model to code correspondance. The sections which you are expected to change are prefixed by the following line:

```
// please fill in/modify the following section
```

The code generator will generate two additional files with sections to preserve in the directory of your model file. The file `undumped.ce` is used to write sections of model elements which were but are no longer assigned to a currently generated module. `deleted.ce` is used to write sections of no longer existing model elements.

The following listing shows a commented module body.

```

// documentation of module
// ... a short comment from the module documentation

// -beg- preserve=no 37EB5DAC02F3 StdIncludes "ex1"
// disable warning C4786: symbol greater than 255 character,
#pragma warning(disable: 4786)
#include <algorithm>
#include <stdexcept>
// ... the StdIncludes property of the module
// -end- 37EB5DAC02F3 StdIncludes "ex1"

// -beg- preserve=no 37EB5DAC02F3 Declarations "ex1"
// ... the declaration tab of the module specification dialog
// -end- 37EB5DAC02F3 Declarations "ex1"

// -beg- preserve=no 37EB5DAC02F3 dependencies "ex1"
// ... dependency relationships of the module
// -end- 37EB5DAC02F3 dependencies "ex1"

// declare something only in the code
// please fill in/modify the following section
// -beg- preserve=no 37EB5DAC02F3 coderDecls "ex1"
// ... you may add your include statements here
// -end- 37EB5DAC02F3 coderDecls "ex1"

#undef EHI_FILE
#define EHI_FILE ehi_file
static const char *ehi_file=__FILE__;

static const char *ehi_class="";
// ... trace from executable to source

// A          documentation of class
// ... each class listed with a short comment
class A;
// ... for each class a forward declaration

// A          documentation of class
// ops        documenation of operation
// attr       documentation of attribute
// ... for each class feature a short comment
class A {
private:
    static const char *ehi_class;

```

```

    // declare/define something only in the code
    // please fill in/modify the following section
    // -beg- preserve=no 37EB5D5B0274 coderDecls "A"
    // ... add your code only declaration to a class here
    // -end- 37EB5D5B0274 coderDecls "A"
public:
    // -beg- preserve=no 37EB601501A6 decl "ops"
    void ops();
    // -end- 37EB601501A6 decl "ops"
private:
    // -beg- preserve=no 37EB5FEF0101 code "attr"
    int attr;
    // -end- 37EB5FEF0101 code "attr"
};

// little helpers (with no model equivalent)
// please fill in/modify the following section
// -beg- preserve=no 37EB5DAC02F3 utilities "ex1"
// ... add utilities belonging to the module here
// -end- 37EB5DAC02F3 utilities "ex1"

// documentation of operation
// -beg- preserve=no 37EB601501A6 code "A::ops"
void A::ops()
// -end- 37EB601501A6 code "A::ops"
{
    const char *ehi_method = "ops";
    // ... trace from executable to source
    // please fill in/modify the following section
    // -beg- preserve=no 37EB601501A6 codebody "A::ops"
    // ... add your instructions here
    return;
    // -end- 37EB601501A6 codebody "A::ops"
}

const char *A::ehi_class="A";
// ... trace from executable to source

// declare/define something only in the code
// please fill in/modify the following section
// -beg- preserve=no 37EB5D5B0274 detail_impl "A"
// ... add your definitions of code local class features here
// -end- 37EB5D5B0274 detail_impl "A"

// main(), globals and other test stuff only in the code
// please fill in/modify the following section

```

```
// -beg- preserve=no 37EB5DAC02F3 standalone "ex1"  
// ... add your code local definitions belonging to the module here  
// -end- 37EB5DAC02F3 standalone "ex1"
```

5 Code generation properties

This section describes all properties used by the code generator. It is meant as a reference and not a guide.

5.1 Macros

In definition text of some properties you may use macros (see for example 5.11.2). In the generated text the macro is replaced by the corresponding model value.

To use a macro, enclose the macroname with curly braces and prefix it with the dollar symbol e.g. `#{oppRoleCodeName}`. You may use some options, appended with a colon to the macro name, to change the generated text e.g. `#{oppRoleCodeName:u}`. Valid options are

- l** make all characters lowercase
- u** make all characters uppercase

5.2 Project

5.2.1 Directory

This property determines the root directory of the generated code files. If it is empty (default), it is the directory of the model file.

5.3 Class

If you set the class type in the specification dialog to `ClassUtility` instead of `Class` some properties may have no or different meaning. There is a comment at these properties.

5.3.1 GenerateDefaultConstructor

This property determines how the default constructor `aClass::aClass()` is generated. See also 5.6.5.

DeclareAndDefine Default. A default constructor is declared in the class specification and defined.

DeclareOnly The default constructor is only declared but not defined. This is useful to prevent the compiler from generating (and using) a wrong one.

DoNotDeclare No default constructor is declared. The compiler will generate one.

This property is not used in a class utility.

5.3.2 DefaultConstructorVisibility

Determines how the default constructor is accessible to other classes.

Public Default. Same as in the C++ language.

Protected Same as in the C++ language.

Private Same as in the C++ language.

This property is not used in a class utility.

5.3.3 skipConstrInit

List of comma separated members which should not be set in the init section of the constructor. This property is not used in a class utility.

5.3.4 GenerateCopyConstructor

This property determines how the copy constructor

```
aclass::aclass(const aclass &);
```

is generated.

DeclareAndDefine A copy constructor is declared in the class specification and defined.

DeclareOnly Default. The copy constructor is only declared but not defined. This is useful to prevent the compiler from generating (and using) a wrong one.

DoNotDeclare No copy constructor is declared. The compiler will generate one.

This property is not used in a class utility.

5.3.5 CopyConstructorVisibility

Determines how the copy constructor is accessible to other classes.

Public Same as in the C++ language.

Protected Same as in the C++ language.

Private Default. Same as in the C++ language.

This property is not used in a class utility.

5.3.6 GenerateDestructor

This property determines if the destructor `aclass::~~aclass()` is generated.

True Default. Generates the destructor.

False Doesn't generate the destructor.

This property is not used in a class utility.

5.3.7 DestructorVisibility

Determines how the destructor is accessible to other classes.

Public Default. Same as in the C++ language.

Protected Same as in the C++ language.

Private Same as in the C++ language.

This property is not used in a class utility.

5.3.8 DestructorKind

Common The destructor can't be overridden in derived classes.

Virtual Default. To allow and ensure proper cleanup in derived classes.

Abstract A destructor must be implemented in derived classes.

This property is not used in a class utility.

5.3.9 GenerateAssignmentOperation

This property determines how the assignment operation

```
aclass & operator=(const aclass &);
```

is generated.

DeclareAndDefine An assignment operation is declared in the class specification and defined.

DeclareOnly Default. The assignment operation is only declared but not defined. This is useful to prevent the compiler from generating (and using) a wrong one.

DoNotDeclare No assignment operation is declared. The compiler will generate one.

This property is not used in a class utility.

5.3.10 AssignmentVisibility

Determines how the assignment operation is accessible to other classes.

Public Same as in the C++ language.

Protected Same as in the C++ language.

Private Default. Same as in the C++ language.

This property is not used in a class utility.

5.3.11 srcTool

Backlink to origin if this class was generated by a tool (script). The string should be in the form: `tool {uniqueids}`. If this value is empty (default), the class was generated by the user.

5.4 Module-Spec

5.4.1 CodeName

This property defines the name of the generated file. If it is not set (default), the file name is the module name with `.h` appended.

5.4.2 StdIncludes

This property defines the headers which most module specifications require. Default value:

```
// disable warning C4786: symbol greater than 255 character,  
#pragma warning(disable: 4786)  
#include <list>  
#include <vector>  
#include <map>
```

5.4.3 srcTool

Backlink to origin if this module was generated by a tool (script). The string should be in the form: `tool {uniqueids}`. If this value is empty (default), the module was generated by the user.

5.5 Module-Body

5.5.1 CodeName

This property defines the name of the generated file. If it is not set (default), the file name is the module name with `.cxx` appended.

5.5.2 StdIncludes

This property defines the headers which most module bodies requires. Default value:

```
// disable warning C4786: symbol greater than 255 character,  
#pragma warning(disable: 4786)  
#include <algorithm>  
#include <stdexcept>
```

5.5.3 srcTool

Backlink to origin if this module was generated by a tool (script). The string should be in the form: `tool {uniqueids}`. If this value is empty (default), the module was generated by the user.

5.6 Operation

5.6.1 OperationKind

Common Default. Generates e.g. `op()`;

Virtual Generates e.g. `virtual op()`;

Abstract Generates e.g. `virtual op()=0`;

Static Generates e.g. `static op()`;

Friend Generates e.g. `friend op()`;

If this operation is a member of a utility class (a class with the `type` field in the specification dialog set to `ClassUtility`), only `Common`, `Static` and `Friend` are valid values.

5.6.2 OperationIsConst

This property determines if the operation inspects (rather than mutates) its object. A `const` member function is indicated by a `const` suffix just after the member function's parameter list. Member functions with a `const` suffix are called "const member functions" or "inspectors." Member functions without a `const` suffix are called "non-const member functions" or "mutators."

True The operation does not modify the state of the object. e.g. `op() const`;

False Default. The operation may modify the state of the object. e.g. `op()`;

5.6.3 ReturnKind

This property determines how the operation result is returned.

Default Default. Built-in types are returned by value and user defined types are returned by pointer. e.g. `int getsize();Address *getaddr();`

Pointer All types are returned by pointer. e.g. `int *get();`

Reference All types are returned by reference. e.g. `int &get();`

Value All types are returned by value. e.g. `int get();`

5.6.4 ReturnIsConst

True Return value is const. e.g. `const Address *get();`

False Default. Return value is not const. e.g. `Address *get();`

5.6.5 OperationType

This property determines if this operation is a nonstandard constructor (one with parameters). This property determines which special properties are used and which special sections are generated.

Default Default. This is a normal operation. In a next version of the code generator the stereotype of the operation is used to determine the operation type.

Constructor This is a constructor operation. The operation name is set to the class name by the generator. No return type is generated. Additional code sections are generated (like for the default constructor). This value may not be used in a class utility.

Normal This is a normal (no constructor) operation.

5.6.6 skipConstrInit

Only used if this is a constructor. List of comma separated members which should not be set in the init section of the constructor.

5.6.7 Code

Operation body to be issued by the generator. Useful by Rose tools and add-ins which create operations.

5.6.8 srcTool

Backlink to origin if this operation was generated by a tool (script). The string should be in the form: `tool {uniqueids}`. If this value is empty (default), the operation was generated by the user.

5.7 Parameter

5.7.1 ParameterDirection

This property is used to determine if the parameter should be set when the operation is called or if it is set when the operation returns. Derived from this property is how parameters are passed.

In Default. The parameter should be set by the caller. Built-in types are passed by value. User defined types are passed by const reference. e.g. `op(int a,const Addr &b);`

Out The parameter is set by the operation. No value is expected when the operation is called. Passed by pointer. e.g. `op(int *a,Addr *b);`

InOut The parameter should be set by the caller and is evtl. modified by the operation. Passed by pointer. e.g. `op(int *a,Addr *b);`

5.7.2 srcTool

Backlink to origin if this parameter was generated by a tool (script). The string should be in the form: `tool {uniqueids}`. If this value is empty (default), the parameter was generated by the user.

5.8 Has

5.8.1 srcTool

Backlink to origin if this relation was generated by a tool (script). The string should be in the form: `tool {uniqueids}`. If this value is empty (default), the relation was generated by the user.

5.9 Association

See also 5.11. A class utility may not have an association.

5.9.1 srcTool

Backlink to origin if this association was generated by a tool (script). The string should be in the form: `tool {uniqueids}`. If this value is empty (default), the association was generated by the user.

5.10 Inherit

5.10.1 srcTool

Backlink to origin if this relation was generated by a tool (script). The string should be in the form: `tool {uniqueids}`. If this value is empty (default), the relation was generated by the user.

5.11 Role

See also 5.9.

5.11.1 GenerateStdOperations

This property determines if operations for manipulation of these roles links are generated. Operations to add, query and remove links may be generated.

True Default. Association manipulation operations are generated.

False No operations are generated.

5.11.2 ExDuplicateKey

Exception thrown by association manipulators if you try to add the same key twice.

You may use the macros `oppRoleCodeName` and `oppQualAttrCodeName` in this definition (see 5.1). `oppRoleCodeName` is the name in the generated code of the opposite role. `oppQualAttrCodeName` is the name of the qualifier or empty if not a qualified association.

Default:

```
std::logic_error("${oppRoleCodeName} with duplicate key")
```

5.11.3 ExKeyNotFound

Exception thrown by association manipulators if you try to retrieve a non existing key.

You may use the macros `oppRoleCodeName` and `oppQualAttrCodeName` in this definition (see 5.1). `oppRoleCodeName` is the name in the generated code of the opposite role. `oppQualAttrCodeName` is the name of the qualifier or empty if not a qualified association.

Default:

```
std::logic_error("no ${oppRoleCodeName} with this key found")
```

5.11.4 ExAlreadyAttached

Exception thrown by association manipulators if you try to add an object if there is already one.

You may use the macros `oppRoleCodeName` and `oppQualAttrCodeName` in this definition (see 5.1). `oppRoleCodeName` is the name in the generated code of the opposite role. `oppQualAttrCodeName` is the name of the qualifier or empty if not a qualified association.

Default:

```
std::logic_error("already a ${oppRoleCodeName} attached")
```

5.11.5 ExNothingAttached

Exception thrown by association manipulators if you try to retrieve an object and there is none.

You may use the macros `oppRoleCodeName` and `oppQualAttrCodeName` in this definition (see 5.1). `oppRoleCodeName` is the name in the generated code of the opposite role. `oppQualAttrCodeName` is the name of the qualifier or empty if not a qualified association.

Default:

```
std::logic_error("no ${oppRoleCodeName} attached")
```

5.11.6 srcTool

Backlink to origin if this role was generated by a tool (script). The string should be in the form: `tool {uniqueids}`. If this value is empty (default), the role was generated by the user.

5.12 Attribute

Attributes in a class utility are always at class scope.

5.12.1 AttributeIsConst

True The variable is not modifiable. e.g. `const int val;`

False Default. The variable is modifiable. e.g. `int val;`

5.12.2 srcTool

Backlink to origin if this attribute was generated by a tool (script). The string should be in the form: `tool {uniqueids}`. If this value is empty (default), the attribute was generated by the user.

5.13 Uses

5.13.1 srcTool

Backlink to origin if this relation was generated by a tool (script). The string should be in the form: `tool {uniqueids}`. If this value is empty (default), the relation was generated by the user.

5.14 Subsystem

5.14.1 CodeName

This property defines the name of the generated subdirectory. If it is not set (default), the name is the subsystem name.

5.14.2 `srcTool`

Backlink to origin if this subsystem was generated by a tool (script). The string should be in the form: `tool {uniqueids}`. If this value is empty (default), the subsystem was generated by the user.

5.15 Category

5.15.1 `srcTool`

Backlink to origin if this category was generated by a tool (script). The string should be in the form: `tool {uniqueids}`. If this value is empty (default), the category was generated by the user.

6 Guide to the source of the code generator

The global variables with the prefix `section_` contain the sections to preserve during code generation.

`printsection`, `readsections` and `writeunusedsections` are used to read and write code sections.

The global variables with the prefix `ts_` and the functions `topsortinit`, `addtopsortcond` and `topsort` implement a topological sort. Topological sorting is used to order the generation of class specifications in `sortclasses`.

The `main` function controls the whole code generation process and also the generation of the things at module level. The `generateTheClass` function controls the generation at class level. Most functions have the two parameters `fileid` and `classspec`. The parameter `fileid` is the filehandle for code generation. The parameter `classspec` is true while the function is called while generating a class specification. The `generateAttribute` function generates code for a user defined attribute. The `generateOperation` function generates code for a user defined operation. The `generateRole` function generates code for one end of an association. The `generateClassStdOperations` generates code for standard class operations e.g. constructor or destructor.

7 Files

`readme.txt` short description and pointer to documentation

`license.txt` package license terms

`bin` Tools

`doc` Documentation

`src` Source

`tests` Tests

tests/checked Files to test against

bin/getptydoc.pl perl script to extract documentation from src/cegen.pty

doc/cegen.tex main LaTeX source of this documentation

doc/cegen.txt this file in text format

doc/cegen.pdf this file in Adobe portable document format

doc/cgp.tex extracted documentation from src/cegen.pty (used by main.tex)

doc/ex1.mdl model of ex1.tex

doc/ex1.tex commented module body as latex source (included by main.tex)

src/cegen.ebs code generator

src/cegen.pty rose property file for use with cegen

src/cegen.mnu rose menu file for use with cegen

src/cegen.reg registry entries for use with cegen

tests/assoc.mdl Model to test code generation of associations

tests/ops.mdl Model to test code generation of operations

tests/qassoc.mdl Model to test code generation of qualified associations

tests/checked/assoc.cxx expected module body of assoc.mdl

tests/checked/assoc.h expected module specification of assoc.mdl

tests/checked/ops.cxx expected module body of ops.mdl

tests/checked/ops.h expected specification body of ops.mdl

tests/checked/qassoc.cxx expected module body of qassoc.mdl

tests/checked/qassoc.h expected module specification of qassoc.mdl