

C++ Stereotypes

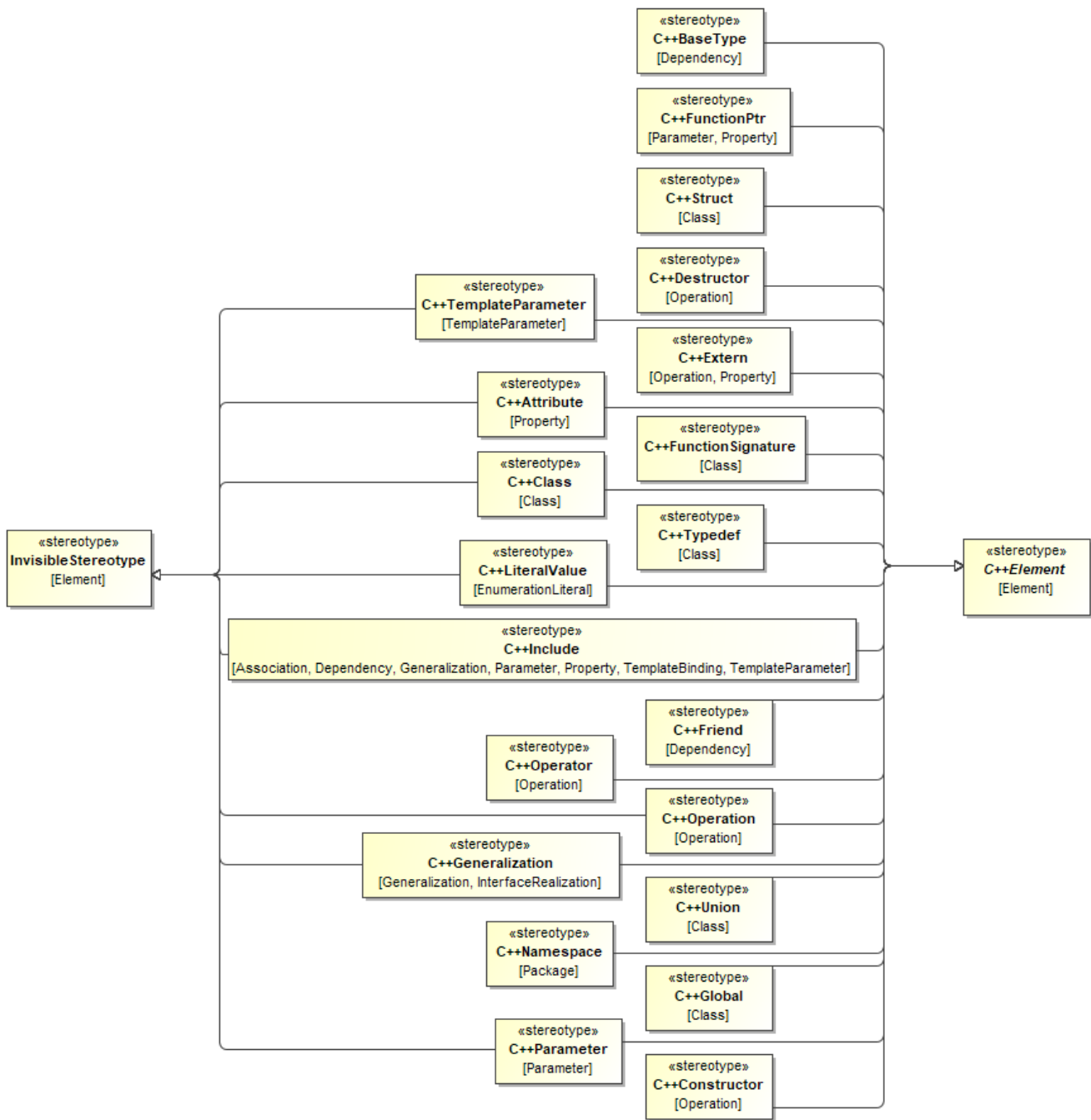
All C++ stereotypes are derived from «C++Element».

Constraints described in this chapter are for information only; syntax of these constraints must be checked with the [Object Constraint Language](#) interpreter.

Invisible stereotypes «*C++Class*», «*C++Operation*», «*C++Parameter*», «*C++Attribute*», «*C++LiteralValue*», «*C++Include*», «*C++Generalization*», and «*C++TemplateParameter*» are used only to store C++ language properties. These stereotypes and their tag definitions are used by the [Domain Specific Language Customization](#) framework.

- C++ Stereotypes

- C++Operation
- C++Operator
- C++Constructor
- C++Destructor
- C++Parameter
- C++ Attribute
- C++LiteralValue
- C++Friend
- C++Struct
- C++Typedef
- C++Union
- C++Global
- C++Namespace
- C++Extern
- C++FunctionPtr
- C++FunctionSignature
- C++Class
- C++BaseType
- C++Include
- C++TemplateParameter
- C++Generalization



C++ Stereotypes

C++Operation

«C++Operation» is an invisible stereotype used to include language properties for any C++ operation.

Name	Meta Class	Constraints
------	------------	-------------

C++Operation	Operation	<p>Function <i>Const</i></p> <pre>void f() const;</pre> <p>Constraint: Only valid for member function (member functions are operators that are declared as members of a class)</p> <p>if isQuery then</p> <p>stereotype-</p> <pre>>select(name='C++Global')->isEmpty()</pre>
Tag definition	Type and default value	Descriptions
inline	boolean[1]=false	<p>Function <i>Inline</i></p> <pre>inline a();</pre>
throw exception	C++ThrowType[1]=any	<p>Exception specification:</p> <p>If operation.raisedExpression is not empty, the expression <i>throw</i> is generated.</p> <pre>void f() throw(int);</pre> <p>If operation.raisedExpression is empty and expression <i>throw</i> value is none, then a expression <i>throw</i> without argument is generated.</p> <pre>void f() throw ();</pre> <p>If operation.raisedExpression is empty and expression <i>throw</i> value is <i>any</i>, it does not generate a Keyword <i>throw</i>.</p> <pre>void f();</pre>
virtual	boolean[1]=false	<p>Function <i>Virtual</i></p> <pre>virtual a();</pre> <p>Constraint: Only valid for member <i>function</i> and <i>non static</i></p> <p>stereotype-</p> <pre>>select(name='C++Global') ->isEmpty() and IsStatic = false</pre>
volatile	boolean[1]=false	<p>Function <i>Volatile</i></p> <pre>void f() volatile;</pre>
functonTryBlock	boolean[1]=false	<p>Function <i>try block</i></p> <pre>void f() try{</pre>

C++Operator

Stereotype «C++Operator» is used to define a C++ operator function. This stereotype extends the stereotype «C++Operation».

Name	Meta Class	Constraints
C++Operator	Operation	<p>Function Operator</p> <pre>T& operator+(T& a);</pre> <p>Constraint: the name start with word <i>operator</i></p>

C++Constructor

Stereotype «*C++Constructor*» is used to define C++ Constructor. This stereotype extends the stereotype «*C++Operation*».

Name	Meta class	Constraints
C++Constructor	Operation	name = owner.name
Tag definition	Type and default value	Description
explicit	boolean[1]=false	Constructor <i>Explicit</i> explicit a();
initialization list	String[0..1]	Constructor initialization: a() : x(1) {}

C++Destructor

Stereotype «*C++Destructor*» is used to define C++ destructor. This stereotype extends Stereotype «*C++Operation*».

Name	Meta Class	Constraints
C++Destructor	Operation	name = "~"+owner.name

C++Parameter

«*C++Parameter*» is an invisible stereotype used to include language properties for any C++ function parameter.

Name	Meta Class	Constraints
C++Parameter	Parameter	
Tag definition	Type and default value	Description
C Type declaration	boolean[1]=false	Declare parameter's type in C style: void a(enum Day x); C++ style: void a(Day x);
Register	boolean[1]=false	Parameter <i>Register</i> void a(register int x);
Array	String[0..1]	C++ <i>Array</i> definition void a(int x[2][2]);

C++ Attribute

An invisible stereotype «*C++Attribute*» is used to include language properties for any C++ variable.

Name	Meta Class	Constraints
------	------------	-------------

C++Attribute	Property	<p>Constraint for code generation: it is valid to have a default value for any kind of attribute, but it is illegal to initialize a member variable within its definition.</p> <pre>class A { int x = 1; };</pre> <p>if defaultValue.size() > 0 then</p> <p>owner.stereotype-</p> <p>>exists(name='C++Global') or (isStatic = true and</p> <p>typeModifiers.contains("const"))</p>
Tag definition	Type and default value	Descriptions
abbreviated initialization	boolean[1]=false	<p>Initialize the attribute with the abbreviated form:</p> <pre>int x(5);</pre> <p>Constraint</p> <p>owner.stereotype-</p> <p>>exists(name='C++Global')</p>
bit field	String[0..1]	<p><i>Bit field</i></p> <pre>int x:2;</pre> <p>Constraint: Only valid for member function</p> <p>stereotype-</p> <p>>select(name='C++Global') ->isEmpty()</p>
c type declaration	boolean[1]=false	<p>Declare attribute's type in C style</p> <p>C style:</p> <pre>enum Day x;</pre> <p>C++ style:</p> <pre>Day x;</pre>
container	String[0..1]	<p>container of the attribute. \$ character is replaced by the attribute type.</p> <pre>vector<\$> x;</pre>
mutable	boolean[1]=false	<p>Attribute mutable modifier.</p> <pre>mutable int x;</pre> <p>Constraint: Only valid for member function</p> <p>stereotype->select(name='C++Global')</p> <p>->isEmpty()</p>
array	String[0..1]	<p>C++ array definition</p> <pre>int x[2][2];</pre>

C++LiteralValue

An invisible stereotype «*C++LiteralValue*» is used to include language properties for any C++ enum field.

Name	Meta Class	Constraints
C++LiteralValue	EnumerationLiteral	
Tag definition	Type and default value	Descriptions

value	String[0..1]	Value definition of an enum field: (A valid C++ expression) <code>enum Day {Mon = 2};</code>
-------	--------------	--

C++Friend

Stereotype «*C++Friend*» is used to define C++ friend relationship.

Name	Meta Class	Constraints
C++Friend	Dependency	Client is Class or Operation and supplier is Class <code>(client.oclIsTypeOf(Class) or client.oclIsTypeOf(Operation)) and supplier.oclIsTypeOf(Class)</code>

C++Struct

Stereotype «*C++Struct*» is used to define C++ struct.

Name	Meta Class
C++Struct	Class

C++Typedef

Stereotype «*C++Typedef*» is used to define C++ typedef.

Name	Meta Class	Constraints
C++Typedef	Class	A typedef does not contain an operation and attribute <code>feature->isEmpty()</code> A «C++BaseType» dependency is defined

C++Union

Stereotype «*C++Union*» is used to define C++ union.

Name	Meta Class
C++Union	Class

C++Global

Stereotype «*C++Global*» is used to define global functions and variables (functions and variables outside a class/struct/union declaration).

Name	Meta Class	Constraints
C++Global	Class	Only one «C++Global» class into a package <code>owner.ownedElement->select(stereotype->select(name='C++Global')).size()=1</code> All operations and attributes are public <code>feature->forAll(visibility = #public)</code>

C++Namespace

Stereotype «*C++Namespace*» is used to define a C++ namespace.

Name	Meta class	
C++Namespace	Package	
Tag definition	Type and default value	Description
unique namespace name	String[0..1]	Unnamed namespace namespace { }

C++Extern

Stereotype «*C++Extern*» is used to define C++ extern variable.

Name	Meta class	Constraints
C++Extern	Operation, Property	<code>owner.stereotype->exists(name='C++Global')</code>
Tag definition	Type and default value	Description
linkage	String[0..1]	Specification <i>Linkage</i> : <code>extern "C"</code>

C++FunctionPtr

Stereotype «*C++FunctionPtr*» is used to define C++ function pointer.

Name	Meta class	
C++FunctionPtr	Parameter, Property	
Tag definition	Type and default value	Description
signature	Operation	The signature of the function (C++ function pointer definition without the operation name)
member class	Class	The class used for pointer to member function.

C++FunctionSignature

Stereotype «*C++FunctionSignature*» is used as a container to model C++ function pointer.

Name	Meta class	Constraints
C++FunctionSignature	Class	The class cannot have properties. <code>properties->isEmpty()</code>

C++Class

Stereotype «*C++Class*» is an invisible stereotype used to include language properties for any C++ variable.

Name	Meta class
C++Class	Class

C++BaseType

Stereotype «*C++BaseType*» is used to link base type of a typedef.

Name	Meta class	Constraints
C++BaseType	Dependency	Client is type of Class with stereotype «C++Typedef».
Tag definition	Type and default value	Description
type modifiers	String[0..1]	Type modifiers of the typedef or function pointer.
member class	Class[0..1]	Memberclass of typedef or function pointer.
array	String[0..1]	Array definition.

C++Include

Stereotype «C++Include» is used to keep information about the include type, for generating include and forward class declaration.

Name	Meta Class	Constraints
C++Include	Association, Dependency, Generalization, Parameter, Property, TemplateBinding, TemplateParameter	Client is type of Component
Tag definition	Type and default value	Description
header include	String	The value of the tag can be one of the following <ul style="list-style-type: none"> • None • User Include • System Include • Class Forward
implementation include	String	The value of the tag can be one of the following <ul style="list-style-type: none"> • None • User Include • System Include • Class Forward

The *header include* tag is used when the client component has header file extension, "*.h"; otherwise, the tag *implementation include* will be used.

The *User Include* tag value is used for generating user include, such as #include "test.h".

The *System Include* tag value is used for generating system include, such as #include <string.h>

The *Class Forward* tag value is used for generating forward class declaration.

C++TemplateParameter

Stereotype «C++Template Parameter» is used to keep type keyword between class and typename for template parameter declaration.

Name	Meta Class	
C++TemplateParameter	TemplateParameter	
Tag definition	Type and default value	Description
type keyword	C++TemplateTypeKeyword=class	The value of tag is one of the following <ul style="list-style-type: none"> • class • typename

C++Generalization

Stereotype «C++Generalization» is used for information related to generalization and interface realization.

Name	Meta Class	

C++Generalization	Generalization, InterfaceRealization	
Tag definition	Type and default value	Description
Inheritance Visibility	C++GeneralizationVisibility[1] = none	<p>The value of tag can be one of the following</p> <ul style="list-style-type: none"> • None • public • protected • private <p>Example: class A : private B {};</p>
Virtual Inheritance	boolean[1]=false	

Related pages

- [Code Engineering](#)
 - [Code Engineering Sets](#)
 - [Generating Code](#)
 - [Reverse Options](#)
 - [Global options for Code Engineering](#)
 - [Files of Properties](#)
 - [Java Code Engineering](#)
 - [C++ Code Engineering](#)
 - [C# Code Engineering](#)
 - [CORBA IDL Mapping To UML](#)
 - [WSDL](#)