AspectC++ Quick Reference

Syntax Overview

The AspectC++ syntax is an extension to the C++ syntax defined in the ISO/IEC 14882:1998(E) standard.

class-head:
`aspect` *identifier*$_{opt}$ *base-clause*$_{opt}$

declaration:
pointcut-declaration
advice-declaration

member-declaration:
pointcut-declaration
advice-declaration

pointcut-declaration:
`pointcut` *declaration*

pointcut-expression:
constant-expression

advice-declaration:
`advice` *pointcut-expression* **`:`** *declaration*

Concepts

aspect
Aspects in AspectC++ implement in a modular way crosscutting concerns and are an extension to the class concept of C++. Additionally to attributes and methods, aspects may also contain *advice declarations*.

advice declaration
An advice declaration is used either to specify code that should run when the *join points* specified by a *pointcut expression* are reached or to introduce a new method, attribute, or type to all *join points* specified by a *pointcut expression*.

join point
In AspectC++ join points are defined as points in the component code where aspects can interfere. A join point refers to a method, an attribute, a type (class, struct, or union), an object, or a point from which a join point is accessed.

pointcut
A pointcut is a set of join points described by a *pointcut expression.*

pointcut expression
Pointcut expressions are composed from *match expressions* used to find a set of join points, from pointcut functions used to filter or map specific join points from a pointcut, and from algebraic operators used to combine pointcuts.

match expression
Match expressions are strings containing a search pattern.

Aspects

Writing aspects works very similar to writing C++ class definitions.

aspect *A* **{ ... };**
defines the aspect *A*
aspect *A* **:** *public B* **{ ... };**
A inherits from class or aspect *B*

Advice Declarations

before(...)
the advice code is executed before the join points in the pointcut
after(...)
the advice code is executed after the join points in the pointcut
around(...)
the advice code is executed in place of the join points in the pointcut

If the advice is not recognized as being of a predefined kind (i.e. **before**, **after**, or **around**), it is regarded as an introduction of a new method, attribute, or type to all join points in the pointcut.

thisJoinPoint
object of type *JoinPoint* to be used by advice code to obtain more information about the current join point.

Pointcut Expressions

Type Matching

`"int"`
matches the C++ built-in scalar type `int`
`"% *"`
matches pointers to any class or named C++ data type

Namespace and Class Matching

`"Chain"`
matches the class, struct or union *Chain*
`"Memory%"`
matches any class, struct or union whose name starts with “Memory”

Attribute Matching

`"Chain* Chain::next"`
matches the attribute *next* of class *Chain* having type `Chain*` (pointer to *Chain*)
`"% Chain::%"`
matches any attribute of class *Chain*

Function Matching

`"void reset()"`
matches the function *reset* having no parameters and returning `void`
`"% printf(...)"`
matches the function *printf* having any number of parameters and returns any type
`"void %(int,%)"`
matches any function having exactly two parameters (from which the first one must be `int`) and returning `void`

Predefined Pointcut Functions

Types

base(*pointcut*) $\mathrm{N}\rightarrow\mathrm{N}_{C,F}$
returns all base classes resp. redefined functions of classes in the *pointcut*
derived(*pointcut*) $\mathrm{N}\rightarrow\mathrm{N}_{C,F}$
returns all classes in the *pointcut* and all classes derived from them resp. all redefined functions of derived classes

Control Flow

cflow(*pointcut*) N→C
captures join points occuring in the dynamic execution context of join points in the *pointcut*

Scope

within(*pointcut*) N→C
filters all join points that are within the functions[†] or classes in the *pointcut*

Functions

call(*pointcut*) $N \to C_C$[‡‡]
provides all join points where a named entity in the *pointcut* is called. *pointcut* may contain function names or class names. In the case of a class name all calls to functions of that class are provided.

execution(*pointcut*) $N \to C_E$
provides all join points referring to the implementation of a named entity in the *pointcut*. *pointcut* may contain function names or class names. In the case of a class name all implementations of functions of that class are provided.

Attributes

set(*pointcut*)[†] $N \to C_S$
selects all join points where the value of an attribute or global variable is modified[‡]

get(*pointcut*)[†] $N \to C_G$
selects all join points where the value of an attribute or global variable is read[‡]

Context

that(*type pattern*) $N \to C$
returns all join points where the current C++ `this` pointer refers to an object which is an instance of a type that is compatible to the type described by the *type pattern*

target(*type pattern*) $N \to C$
returns all join points where the target object of a call is an instance of a type that is compatible to the type described by the *type pattern*

result(*type pattern*)[†] $N \to C$
returns all join points where the result object of a call is an instance of a type that is compatible to the type described by the *type pattern*

args(*type pattern*, ...) $(N,...) \to C$
receives a list of *type patterns* and filters all methods or attributes with a matching signature

Instead of the *type pattern* it is possible here to deliver the name of a variable to which the context information is bound. In this case the type of the variable is used for the type matching.

Algebraic Operators

pointcut **&&** *pointcut* $(N,N) \to N, (C,C) \to C$
intersection of the join points in the *pointcuts*

pointcut || *pointcut* $(N,N) \to N, (C,C) \to C$
union of the join points in the *pointcuts*

! *pointcut* $N \to N, C \to C$
exclusion of the join points in the *pointcut*

JoinPoint-API

Types

Result
result type of a function

That
object type (object initiating a call)

Target
target object type (target object of a call)

Functions

static AC::Type **type**()
returns the encoded type for the join point conforming with the C++ ABI V3 specification[††]

static int **args**()
returns the number of arguments of a function for call and execution join points

static AC::Type **argtype**(*int number*)
returns the encoded type of an argument conforming with the C++ ABI V3 specification[††]

static const char ***signature**()
gives a textual description of the join point (function name, class name, ...)

static unsigned int **id**()
returns a unique numeric identifier for this join point

static AC::Type **resulttype**()
returns the encoded type of the result type conforming with the C++ ABI V3 specification[††]

static AC::JPType **jptype**()
returns a unique identifier describing the type of the join point

void ***arg**(*int number*)
returns a pointer to the memory position holding the argument value with index *number*

Result ***result**()
returns a pointer to the memory location designated for the result value or 0 if the function has no result value

That ***that**()
returns a pointer to the object initiating a call or 0 if it is a static method or a global function

Target ***target**()
returns a pointer to the object that is the target of a call or 0 if it is a static method or a global function

void **proceed**()[†]
executes the original join point code in an around advice

AC::Action &**action**()
returns the runtime action object containing the execution environment to execute (*trigger()*) the original functionality encapsulated by an around advice

Example

A reusable tracing aspect.

```
aspect Trace {
  pointcut virtual functions() = 0;
  advice execution(functions()) : around() {
    cout << "before " << JoinPoint::signature() << "(";
    for (unsigned i = 0; i < JoinPoint::args(); i++)
      cout << (i ? ", " : "") << JoinPoint::argtype(i);
    cout << ")" << endl;
    thisJoinPoint->action().trigger();
    cout << "after" << endl;
  }
};
```

In a derived aspect the pointcut *functions* may be redefined to apply the aspect to the desired set of functions.

```
aspect TraceMain : public Trace {
  pointcut functions() = "% main(...)";
};
```

This is a reference sheet corresponding to AspectC++ 0.6.
Version 1.0, 5th February 2003.

[†] not yet implemented in version 0.6

[‡] does not recognize access through C++ references or pointers

[††] `http://www.codesourcery.com/cxx-abi/abi.html#mangling`

[‡‡] C, C_C, C_E, C_S, C_G: Code (any, only *Call*, only *Execution*, only *Set*, only *Get*); N, N_N, N_C, N_F, N_T: Names (any, only *Namespace*, only *Class*, only *Function*, only *Type*)