Paper Number	P0329R1
Date	2016-09-26
Authors	Tim Shen <timshen@google.com> Richard Smith <richard@metafoo.co.uk>
Audience	CWG

Designated Initialization Wording

This is a formal wording for the designated initialization proposal P0329R0.

Wording

Change 8.6 [dcl.init]p1 as follows

braced-init-list:
{ initializer-list ,opt }
{ designated-initializer-list ,opt }
{ }

designated-initializer-list:
designated-initializer-clause
designated-initializer-list , designated-initializer-clause

designated-initializer-clause:
designator brace-or-equal-initializer

designator:
. identifier

Change in 8.6.4 [dcl.init.list]p1:

List-initialization is initialization of an object or reference from a *braced-init-list*. Such an initializer is called an *initializer list*, and the comma-separated *initializer-clause*s of the *initializer-list* ~~list~~ or *designated-initializer-clause*s of the *designated-initializer-list* are called the *elements* of the initializer list. [...]

Add a new bullet at the start of 8.6.4 [dcl.init.list]p3:

If the *braced-init-list* contains a *designated-initializer-list*, `T` shall be an aggregate class whose non-static data members include the *identifier*s of the *designated-initializer-clause*s of the *designated-initializer-list* in declaration order, and aggregate initialization is performed ([dcl.init.aggr]).

Add a new paragraph to 8.6.1 [dcl.init.aggr]:

The initializations of the elements of the aggregate are evaluated in the element order. That is, every value computation and side effect associated with a given element is sequenced before every value computation and side effect associated with any element that follows it in order.

Change in 8.6.1 [dcl.init.aggr]p3:

When an aggregate is initialized by an initializer list as specified in 8.6.4, the elements of the initializer list are taken as initializers for the elements of the aggregate. If the initializer list is a *designated-initializer-list*, the aggregate shall be of class type, and the initialized elements of the aggregate are the elements named by the *identifier*s in each *designated-initializer-clause*, where each *identifier* shall name a direct non-static data member of the class. Otherwise, the initialized elements of the aggregate are the first *n* elements of the aggregate, in order, where *n* is the number of elements in the initializer list, excluding the second and subsequent non-static data member of a union. Each initialized element is copy-initialized from the corresponding *initializer-clause* or the *brace-or-equal-initializer* of the corresponding *designated-initializer-clause*. If ~~the initializer-clause is an expression~~ that initializer is of the form *assignment-expression* or = *assignment-expression* and a narrowing conversion (8.6.4) is required to convert the expression, the program is ill-formed. [Note: If an initializer~~-clause~~ is itself an initializer list, the element is list-initialized, which will result in a recursive application of the rules in this section if the element is an aggregate. — end note] [Example: …]

Change 8.6.1 [dcl.init.aggr]p6 as follows

[Note: Static data members, anonymous union members, and anonymous bit-fields are not considered elements ~~members~~ of the class for purposes of aggregate initialization. — end note]

Change 8.6.1 [dcl.init.aggr]p7 as follows

An ~~*initializer-list*~~ initializer list is ill-formed if the number of *initializer-clauses* or *designated-initializer-clauses* exceeds the number of ~~members or~~ elements ~~to initialize~~ of the aggregate, if multiple *designated-initializer-clause*s name the same element, or if multiple members of the same union are initialized elements.

Change 8.6.1 [dcl.init.aggr]p8 as follows

~~If there are fewer initializer-clauses in the list than there are elements in the aggregate, then~~ Each non-variant element of the aggregate that is not ~~explicitly~~ an initialized element is ~~shall be~~ initialized from its default member initializer (9.2) or, if there is no default member initializer, from an empty initializer list (8.6.4). If the aggregate is a union, or for each anonymous union member of a non-union aggregate of class type, if no union member is an initialized element, then:

- If any union member has a default member initializer, that member is initialized from its default member initializer.
- Otherwise, the first member of the union (if any) is copy-list-initialized from an empty initializer list.

Change 8.6.1 [dcl.init.aggr]p11 as follows

If an incomplete or empty ~~*initializer-list*~~ initializer list leaves a member of reference type uninitialized, the program is ill-formed.

Change 8.6.1 [dcl.init.aggr]p17 as follows

[Note: When a union is initialized with a brace-enclosed initializer, only one non-static data member can be initialized. ~~the braces shall only contain an initializer-clause for the first non-static data member of the union~~. [Example:

```
union u { int a; const char* b; };
   u a = { 1 };
   u b = a;
   u c = 1; // error
   u d = { 0, "asdf" }; // error
   u e = { "asdf" }; // error
   u f = { .b = "asdf" };
   u g = { .a = 1, .b = "asdf" }; // error
```

] — end note]

Add new paragraph after 13.3.3.1.5 [over.init.list]p1 as follows

If the initializer list is a *designated-initializer-list*, a conversion is only possible if the parameter has an aggregate type that can be initialized from the initializer list according to the rules for aggregate initialization ([dcl.init.aggr]), in which case the implicit conversion sequence is a user-defined conversion sequence whose the second standard conversion sequence is an identity conversion. [Example:

```
struct A { int x, y; };
struct B { int y, x; };
void f(A a, int);  // #1
void f(B b, …);  // #2
void g() {
   f({.x = 1, .y = 2}, 0);  // OK, calls #1
   f({.y = 2, .x = 1}, 0);  // error, selects #1, initialization of a fails
                            // due to [dcl.init.list]p3

}
```

— end example]