Paper Number	P0329R2
Date	2016-11-21
Authors	Tim Shen <timshen@google.com> Richard Smith <richard@metafoo.co.uk>
Audience	CWG

P0329R2: 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

Add a new paragraph as 8.6 [dcl.init]p20:

The same *identifier* shall not appear in multiple *designator*s of a *designated-initializer-list*.

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 where the names of the non-static direct data members of `T` include the *identifier*s of the *designated-initializer-clause*s of the *designated-initializer-list* and those members are declared in the same order as the corresponding *designated-initializer-clause*s. Aggregate initialization is performed ([dcl.init.aggr]). [Example:

```
  struct A { int x; int y; }
  void f() {
    A a{.y = 2, .x = 1};   // error; designator order does not match declaration order
  }
```

]

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, all value computations and side effects associated with a given element are sequenced before those of any element that follows it in order.

Drafting note: unlike 8.6.4/4, this also covers the initialization of elements for which no initializer is explicitly provided.

Change in 8.6.1 [dcl.init.aggr]p3 and split it into two paragraphs:

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. The *explicitly initialized elements* of the aggregate are determined as follows:

- If the initializer list is a *designated-initializer-list*, the aggregate shall be of class type, the *identifier* in each *designated-initializer-clause* shall name a direct non-static data member of the class, and the explicitly initialized elements of the aggregate are the elements that are, or contain, those members.
- If the initializer list is an *initializer-list*, the explicitly 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.
- Otherwise, the initializer list must be `{}`, and there are no explicitly initialized elements.

For each explicitly initialized element:

- If the element is not an anonymous union object, or is initialized by an *initializer-list*, it 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]

- Otherwise, the object is initialized by the *designated-initializer-list* `{ D }`, where *D* is the *designated-initializer-clause* naming a member of the anonymous union object. There shall be only one such *designated-initializer-clause*.

[Example: …

```
  struct A {
    int a;
    string b;
  };
```

`A{.b{"a"}}` has the following steps:

1. Initialize a with {}
2. Initialize b with {"a"}

]

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

[Note: Static data members, non-static data members of 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* is ill-formed if the number of *initializer-clauses* exceeds the number of ~~members or~~ elements ~~to initialize~~ of the aggregate.

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 element that is not an explicitly initialized element 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).
[Example: ...

```
  struct A {
    string a;
    int b = 42;
    int c = -1;
  };
```

`A{.c=21}` has the following steps:

1. Initialize a with {}
2. Initialize b with = 42
3. Initialize c with = 21

]

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

When a union is initialized with an ~~brace-enclosed~~ initializer list, there shall not be more than one explicitly initialized element. ~~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
```

]

Add new paragraph after 13.3.3.1.5 [over.ics.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 second standard conversion sequence is an identity conversion. [Note: Aggregate initialization does not require that the members are declared in designation order. If, after overload resolution, the order does not match for the selected overload, the initialization of the parameter will be ill-formed ([dcl.init.list]). [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 non-matching member order ([dcl.init.list])
}
```

— end example] — end note]

Change 13.3.3.1.5 [over.ics.list]p2 as follows

Otherwise, if the parameter type is an aggregate [...]