Object Definition

- Example:

```
class Mytype
{
  int x1;
  D0_CLASS_SETUP (Mytype);
};

class Obj : d0_Object
{
  int a;
  d0_String b;
  d0_Ref<Obj> c;
  d0_Vector<float> d;
  Mytype f;
  double g;  // ! This field is not persistent.
  D0_OBJECT_SETUP (Obj);
};
```

Persistent Data Members

- Fundamental types: `int`, `short`, `char`, `bool`, `float`, `double`; `d0_Int`, `d0_UInt`, `d0_Short`, `d0_UShort`, `d0_Char`, `d0_Bool`, `d0_Float`, `d0_Double`, `d0_Octet`.
- `string` and `d0_String`.
- Enumeration types.
- Previously declared structures or classs.
- References.
- Collections.

Collections

- Available:
  - `d0_List<T>`
  - `d0_Vector<T>`
  - `d0_Set<T>`
  - `d0_Multiset<T>`
  - `d0_Map<K, T>`
  - `d0_Multimap<K, T>`
- All derive from the corresponding STL class.
- All also derive from `d0_Collection<T>`, which provides common ODMG-style operations.
- ODMG-style `d0_Iterator<T>`, created by `d0_begin` and `d0_end` methods. Also `d0_Const_Iterator<T>`.

Restrictions

- No multiple inheritance.
- No nested structure definitions.
- No templates except for predefined collection and reference types.
- No C++ pointers or references.
- No bitfields.
- No unions.

References

- Reference type: `d0_Ref<T>`.
- `T` must derive from `d0_Object`.
- Provides:
  - Deferred reads.
  - Reference counting of objects.
- Usage:

```
d0_Ref<Obj> r;
r->func (r->memvar);
Obj tmp = *r;
Obj* p = r.ptr ();
```

Reference Counting

- Objects allocated with 'new' are reference-counted by default.

- Example:

```
d0_Ref<foo> a1 = new foo (1);
// foo(1) now has one reference.
{
  d0_Ref<foo> a2 = a1;
  // now it has two references.
  a1 = new foo (2);
  // now foo(1) has one ref. again.
  a2 = a1;
  // all references to foo(1) are
  //  now gone; foo(1) is deleted.
  //  foo(2) now has two references.
}
// a2 goes away.  foo(2) now has
//  one reference.
a1.clear ();
// now foo(2) is deleted.
```

- Watch out for cycles!

Reference Conversion

- Given

```
class foo : public d0_Object {};
class bar : public foo {};
```

- C++ will default-convert `bar*` to `foo*`.
- But not `d0_Ref<bar>` to `d0_Ref<foo>`.
- Need member templates to do conversion automatically.
- For now use explicit conversions:

```
d0_Ref<foo> rf;
d0_Ref<bar> rb;

rf = D0_REFCONV((foo)) (rb);
  // Default conversions only.
rb = D0_REFCAST((bar)) (rf);
  // With runtime check.
rb = D0_REF_UNSAFECAST((bar)) (rf);
  // No checks.
```

- Special case: `d0_Ref<T>` $\longrightarrow$ `d0_Ref<const T>` done automatically.