```
    bool d_isProduction;
    long d_userId;
    int  d_dataCenterId;
    // ... other information about the context being run in

    static Context* defaultContext();
};
```

Each type that needs `Context` information would take an optional argument to specify a local context; otherwise, it would use the default context:

```
#include <string>  // std::string

struct ContextualObject
{
// ...
     ContextualObject(const std::string& name,
                      Context* context = Context::defaultContext());
// ...
};
```

When combining many objects, all of which might need to access the same context for configuration, it becomes important to pass the context specified at construction to each subobject:

```
struct CompoundObject
{
// ...
    ContextualObject d_o1;
    ContextualObject d_o2;
// ...
    CompoundObject(Context* context = Context::defaultContext())
    : d_o1("First", context)
    , d_o2("Second", context)
    { }
// ...
};
```

This situation might seem well suited for using default member initializers, but the naive approach would have a serious flaw:

```
struct CompoundObject
{
// ...
    ContextualObject d_o1{"first"};   // Bug, does not use context passed to
    ContextualObject d_o2{"second"};  // CompoundObject constructor
// ...
    CompoundObject(Context* context = Context::defaultContext());
// ...
};
```