
```

class CExampleObject: public CMyBaseClass
{
public:
    inline void SetDataMember (lushort value);
    .
    .
private:
    lushort cDataMember;
    .
    .
}
inline void CExampleObject::SetDataMember (lushort value)
{
    CheckVersion();           // call CEngineObject method to check
                             // if we need a new version before the data
                             // member is changed. A New
                             // version will be created, if
                             // necessary
    cDataMember = value;
}

```

As previously mentioned, the base method, `CheckVersion()`, will determine if the existing version of the object is current by comparing the object version to the current document version. If the object version is not current, the `CheckVersion()` method will call a `NewVersion()` method in the object. For example, sample code for the `NewVersion()` method might look like the following code fragment for the above-mentioned object, `CExampleObject`:

```

void CExampleObject::NewVersion (void)
{
    new CExampleObject (this)
}

```

The purpose of `NewVersion()` method is to create a copy of the existing object (identified by the `*this` pointer) by calling a special copy constructor which is discussed below. This “new” object copy then becomes the “old” version of the object while the existing object (`*this`) becomes the “new” version. Since the existing object is the “new” version, upon return from the `CheckVersion()` method, the same modification operations can be performed whether a “new” version was created or not. Therefore, it is not necessary to check whether the `CheckVersion()` method created a new version or not.

Each subclass of `CEngineObject` must declare and implement a “version constructor” which is a copy constructor that copies all members of the current object version. It has the form illustrated by the following code fragment:

```

CExampleObject::CExampleObject (CExampleObject *pOther)
: CMyBaseClass ((CMyBaseClass *) pOther)
{
    cMember1 = pOther->cMember1;
    cMember2 = pOther->cMember2;
    .
    .
}

```

This constructor constructs the “old” object version from the new object version to which the pointer `Other` points. In order to prevent an infinite recursion, the members must be modified directly without calling the `CheckVersion()` method first.

Member objects also present special cases. In particular, when a member object is modified, it is necessary not only

to check and, possibly, create a new version copy for the object modified, but also the object in which the modified object is a member must be checked to insure its version is current. Unfortunately, due to the nature of object-oriented programming, the member object does not automatically have information which would indicate the identity of the containing object or even if it is a member of some object.

In accordance with a preferred embodiment, the member object is provided with information that indicates whether any objects contain it and identifies those objects. The identifying information for each object is stored in an attribute which is part of each object. This attribute is a `CEngineObject` pointer designated as `*cWhole`. When a member object is constructed in a containing object, the member object is passed a pointer to the containing object. The member object saves this pointer in the `*cWhole` attribute. Later, when the member object is about to be changed it calls the `CheckVersion()` method of the containing object via the stored pointer, `cWhole>CheckVersion()` to update the object version of the containing object.

This functionality has been encapsulated in an intermediate class, the `CMember` class. The `CMember` class has a constructor of the form `CMember(CEngineObject*)` and it has an inline `CheckVersion()` which will perform the method call `cWhole>CheckVersion()`. Consequently, any subclasses derived from the `CMember` class will have the capability discussed above. The following code fragment is an example of a derivation of a subclass, `CExampleMember`, from the `CMember` class:

```

class CExampleMember:public CMember
{
public:
    CExampleMember (CEngineObject *pWhole);
    CExampleMember (CEngineObject *pWhole,
                  CExampleMember *pOther);
    void SetData (lushort value);
    .
    .
private:
    lushort cMyData;
    .
    .
};
CExampleMember::CExampleMember (CEngineObject *pWhole)
: CMember (pWhole)
{
    cMyData = 0;
    .
    .
}
CExampleMember::CExampleMember (CEngineObject *pWhole,
    CExampleMember *pOther)
: CMember (pWhole)
{
    cMyData = pOther->cMyData;
    .
    .
}
void CExampleMember::SetData (lushort value)
{
    CheckVersion();           // call CMember::CheckVersion()
    cMyData = value;
}

```

The following code fragment is an example of a subclass, `CExampleObject`, that uses the `CMember` class:

```

class CExampleObject: public CMyBaseClass
{

```
