

Listing One

```
typedef ACE_Task<ACE_MT_SYNCH> Task_Base;
template <class T>
class Task : public Task_Base
{
public:
    Task (const string &taskName);
    // Initialize our Task with a name
    virtual ~Task (void);
    virtual int init();
    // Prevents compiler complaints about hidden virtual functions.
    virtual int shutdown();
    // This closes down the Task and all service threads.
    virtual int put(T *pT);
    virtual T* get();
    // virtual list<Test*> getAll();
    virtual int svc();
    // Actual service loop each of the service threads iterates through.
    virtual void run() = 0;
    static ACE_Lock_Adapter<ACE_SYNCH_MUTEX> *lock_adapter (void);
    // Returns a pointer to the lock adapter.
protected:
    int initTask();
private:
    string m_taskName;
    ACE_Barrier m_barrier;
    // Simple Barrier to make sure all service threads have
    // entered their loop before accepting any messages.
    static ACE_Lock_Adapter<ACE_SYNCH_MUTEX> lock_adapter_;
    // This Lock Adapter is used to synchronize operations
    // on the ACE_Message_Block objects
};
```

Listing Two

```
#ifndef TEST_HPP_
#define TEST_HPP_
#include <map>
typedef enum (kBegin, kBandwidth, kLatency, kEnd) TestType;
/** Result Class */
class Result
{
public:
    Result();
    Result(const Result &result); //copy constructor
    Result& operator= (const Result &result); //assignment operator
    virtual ~Result() {};
    void addResult(double dVal);
    void addResult(long lVal);
    double getResultDouble() (return m_val.m_dVal);
    long getResultLong() (return m_val.m_lVal);
    unsigned long getTestTime() (return m_ulTimeTest);
    unsigned long getRequestTime() (return m_ulTimeRequest);
private:
    union TestValue
    {
        double m_dVal;
        long m_lVal;
    } m_val;
    unsigned long m_ulTimeRequest; //in gmt time
    unsigned long m_ulTimeTest; //in gmt time
};
/** Test Class */
class Test
{
public:
    typedef map<TestType, Result> ResultColl;
    typedef map<TestType, Result>::iterator ResultIter;
public:
    Test() : m_curTest(kBegin), m_ulTarget(0) {};
    Test(Test &test); //copy constructor
    Test& operator=(Test &test); //assignment operator
    ~Test() {};
    unsigned long getTarget() (return m_ulTarget);
    void setTarget(unsigned long ulTarget) (m_ulTarget = ulTarget);
    void addTest(TestType type);
    TestType getActiveTest() (return m_curTest);
    TestType nextTest();
    void setResult(TestType type, long lVal);
    void setResult(TestType type, double dVal);
    void getResult(TestType type, double &dVal);
    void getResult(TestType type, long &lVal);
    ResultColl getResults() (return m_results);
private:
    unsigned long m_ulTarget;
    ResultColl m_results;
    TestType m_curTest;
};
#endif //TEST_HPP_
```

Listing Three

```
void ToolManager::run()
{
    Test *pTest, *pOutTest;
    ToolIter iter;
    ToolBase *pToolBase;
    while (true)
    {
        auto_ptr<Test> pTest(get());
        if (pTest.get() != NULL)
        {
            pToolBase = getTool(pTest->getActiveTest());
            pOutTest = new Test(*pTest);
            pToolBase->put(pOutTest);
        }
    }
}
```

```
    }
}
void ToolManager::CompleteTest::run()
{
    Test *pTest, *pOutTest;
    while (true)
    {
        auto_ptr<Test> pTest(get());
        if (pTest.get() != NULL)
        {
            pOutTest = new Test(*pTest);
            if (pTest->nextTest() == kEnd)
            {
                m_pToolManager->m_pActionManager->put(pOutTest); //done
            }
            else
            {
                put(pOutTest); //put back into my own queue
            }
        }
    }
}
```

DDJ