**BCB6에서 UnitTest 사용하기(1)**

*사전 준비

1. BCB4 버전 이상이 있어야 합니다. (이전 버전 사용자는 UnitTest 버전을 달리해야 합니다) Trial 버전이 필요하신 분은 볼랜드(http://www.borland.com) 이나 한국 볼랜드 커뮤니티인 볼랜드포럼(http://www.borlandforum.com)에 문의하시면 구할 수 있습니다.
2. TDD에 대해서 기본 지식은 있다고 가정합니다. http://www.tdd.or.kr/moin.py 를 참조하시길 바랍니다.

*** UnitTest 받는 방법**

1. http://xprogramming.com/software.htm 로 가셔서
2. "C++ Builder 4.0 and up" 링크를 찾아 CppUnit17BCB30Pro.zip 를 받으시길 바랍니다.

*** UnitTest를 BCB6에서 시도**

1. CppUnit17BCB30Pro.zip의 압축을 풀어서 생기는 CppUnit17BCB30Pro 폴더를 원하는 위치로 옮깁니다.
2. BCB6을 실행하고,
3. File >> Open Project 를 하시고 CppUnit17BCB30Pro 폴더를 살펴 보시고 CppUnit15ProjectGroup.bpg 를 선택합니다.

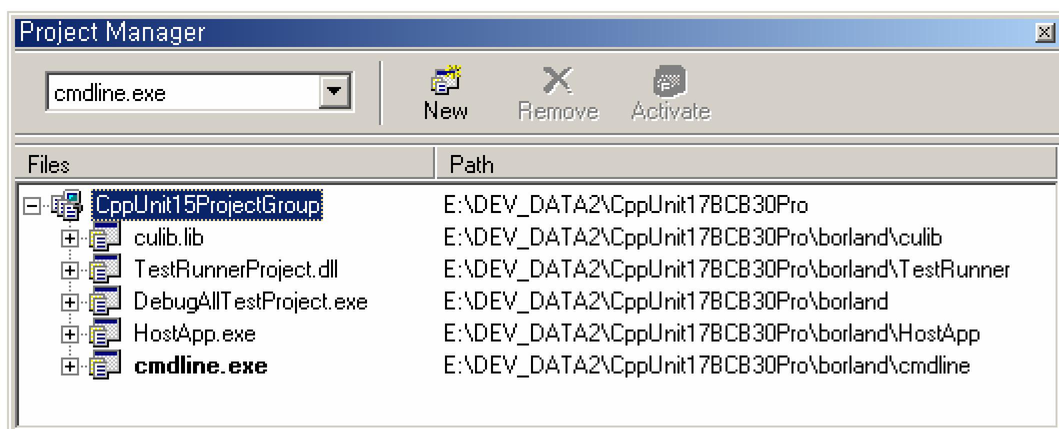


Fig. 1 프로젝트 열기 후 프로젝트 메니져

UnitTest는 다양한 개발 환경에 적용 되어 지도록 많은 포트가 있습니다. 물론 여기에는 자발적으로 노력해준 많은 개발자가 있었지만, 자신의 개발환경에서도 UnitTest를 하고자 하는 욕망이 강했기 때문에 이러한 결과가 있었다고 여겨집니다. 하여튼 포트가 많은 만큼 각 환경에 맞게 개발되다 보니 개발 방법은 다르지만 기본 구조는 동일합니다.

UI 관점에서 봤을 때, GUI를 지원하는 포트와 command line만 지원하는 포트가 있습니다. GUI 지원하는 포트의 대표적인 경우가 JUnit인데요, 이는 대부분의 Java 개발 툴에서 통합환경으로 지원하고 있습니다. JBuilder9는 물론이고 Eclips 에서도 지원을 합니다. 그리고 UnitTest를 좋아하는 개발자가 많은 Python 개발자들은 주로 command line을 위주로 사용합니다. 물론 C 에서도 마찬가지이겠지요.

기본 클래스의 의미만 정리하겠습니다. 자세한 분석은 기회가 또 있을 거라 생각됩니다. 크게 4개의 클래스가 있습니다. 물론 더 많지만 중요한 것 만 추려서 그렇다는 거지요.

- Test
  UnitTest의 기본 클래스. Test result의 run과 collect에 대한 가상함수 포함하고 있습니다.

- TestCase
  UnitTest의 기본 단위 클래스. 실제 테스트 시에 사용 되는 클래스 입니다.

- TestSuite
  TestCase의 패키지 클래스. TestCase를 그룹으로 관리해 줍니다.

- TestResult
  Test의 결과를 처리하는 클래스.

이름으로 유추해봐도 알만한 내용으로만 정리했습니다. 이름만 엮으면 이런 식의 이야기가 나옵니다.

"Test 클래스에서 파생된 TestCase로 UnitTest를 하고, 이 TestCase를 TestSuite로 묶어서 관리하기도 한다. 테스트 결과는 TestResult에서 관리하므로 테스트의 성공 여부를 확인하기 위해서는 TestResult를 잘 살펴야 한다."

이제 본격적으로 UnitTest를 해봅시다. 먼저 미리 제공되는 예제를 살펴본 후에 실제 프로젝트에 어떻게 적용하는 지를 살펴 보지요. 프로젝트 메니져에서 culib.lib 프로젝트를 activate 프로젝트로 합니다. 프로젝트 명 위에서 더블클릭 또는 팝업 메뉴 연 후 activate 선택하면 됩니다. 이제 빌드를 해봅시다. 여기서 빌드는 compile과 linking을 동시에 하는

것을 의미합니다. Warning 정도만 발생하고 Link error 같은 심각한 문제는 발생하지 않을 겁니다. (문제가 발생한다면 곤란합니다. 답해드리기 힘드니까요) 이제 TestRunnerProject.dll 을 동일한 방법으로 activate 프로젝트를 한 후 빌드 합니다. 역시 잘 될 겁니다. 앞의 두 과정을 Run을 하지는 말기 바랍니다. DLL을 위한 프로젝트 이므로 결과물은 .exe가 아니라 .dll, .lib 일 것 입니다. 실수로 Run을 하셨으면 에러창이 나타날 건데요, 무시하셔도 됩니다. DLL이라서 실행이 안 된다는 메시지니까요.

이제 dll이 빌드 되는 것을 확인했으니, DebugAllTestProject를 activate로 하고 빌드를 해 봅니다. 잘되지요? 이 프로젝트가 BCB용 UnitTest에서 제공하는 실행 가능한 샘플 패키지입니다. 사실상 UnitTest의 사용만이 목적이라면 이 프로젝트만 살펴봐도 충분합니다. 얼마나 충분한지 시험해봅시다. 빌드가 문제 없이 되었다면 Run을 해봅니다.

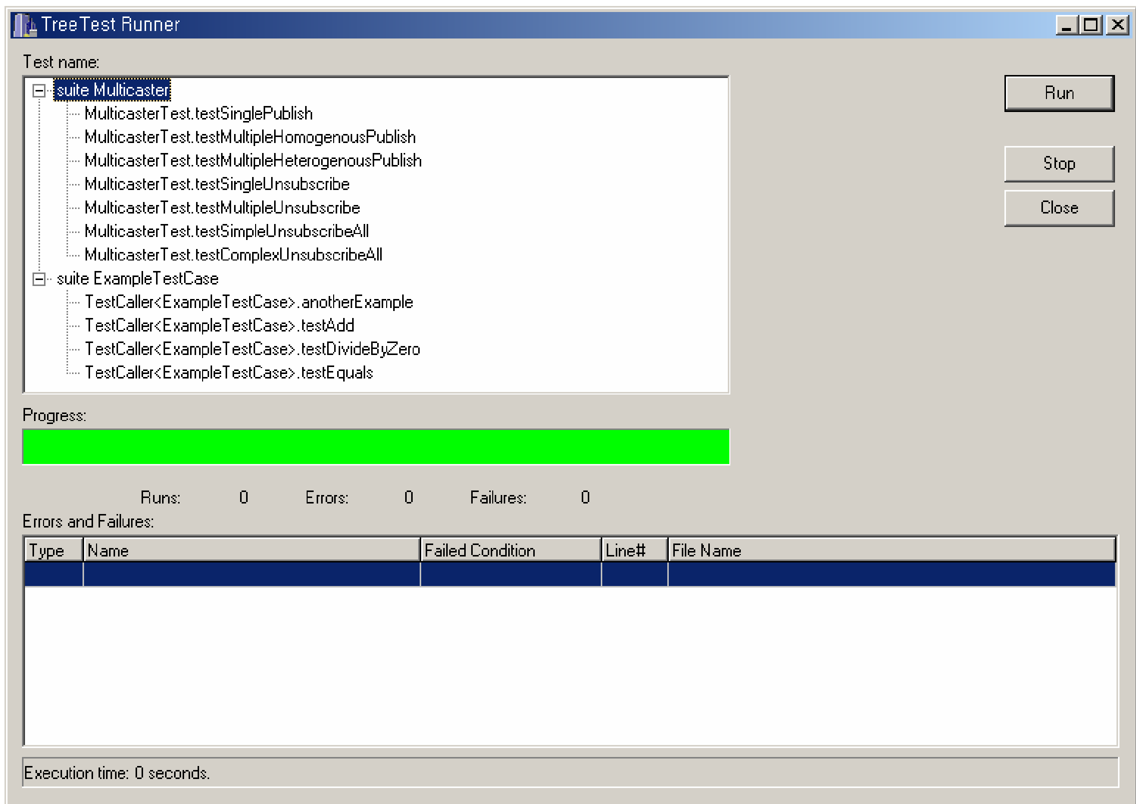


Fig. 2 샘플용 UnitTest 프로젝트 실행

실행을 성공하셨다면 Fig. 2 와 같은 Form이 실행될 것입니다. Suite Multicaster를 선택하고 Run을 선택합니다.

Fig. 3 테스트 성공

Fig. 3 과 같이 Progress Bar에 초록색이 다 차있으면 테스트는 성공한 것입니다. 자 이제 자축하셔도 됩니다. UnitTest를 다 해본 것이니까요. “Hello, World!”를 해본 것이나 다름 없습니다. 이제 활용하는 것만 남았지요. 다음 단계로 suite ExampleTestCase를 선택하고 Run 해봅시다. 아! Run을 하기 전에 BCB6에서 실행을 하고 있는 상태라면, Program Reset(Ctrl+F2) 하시고, 탐색기에서 “%CppUnit17BCB30Pro 상위 폴더%\CppUnit17BCB30Pro\borland\” 에 보시면 빌드 된 DebugAllTestProject.exe 가 있을 겁니다. 이를 실행하셔서 suite ExampleTestCase를 Run 하시길 바랍니다. 이유는 해보시면 알겠지만, BCB6에서 실행 시에 Test 결과가 Fail(Error이 아닙니다. 정확한 의미를 부여하기는 힘들지만, Error는 의도하지 않은 예외상황이고 Fail은 예상할 수 있는 문제라고 해둡시다) 이 되면 throw를 호출하게 되는데요, BCB6가 이를 가져와 버립니다. 원하는 것은 이런 화면을 보는 것인데 말이죠.

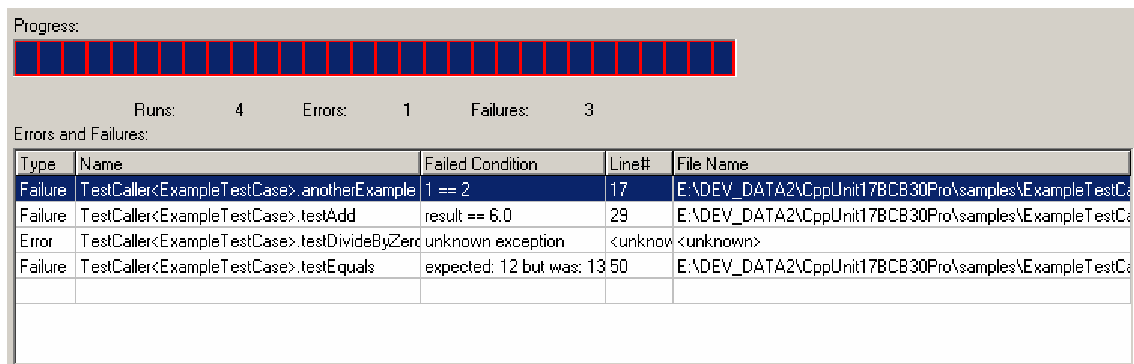


Fig. 4 테스트 Failure

아쉬운 부분은 Error or Failures에 있는 항목을 정보를 가지고 수동으로 문제된 부분을 찾아 나가야 한다는 점입니다. 물론 JUnit을 통합한 JBuilder나 Eclips는 코드로 링크가 되게끔 되어있어서 바로 찾을 수 있습니다. 이제 사용해 볼 수 있는 것은 다 해봤습니다. 남은 것이라고는 각 suite를 확장해서 TestCase의 리스트를 보는 정도 입니다. 이미 해보셨겠죠?

프로젝트에 적용하기 전에 예제로 제공된 환경을 조금 더 살펴 봅시다. UnitTest를 이해하기가 훨씬 수월해 질 것입니다.

* HostAppUnitForm.cpp .h

THostAppForm 클래스를 포함하고 있습니다. 호스트 폼 역할을 하고, 생성시에 suite Test 를 추가 합니다. 그리고 Fig.2 의 테스트 폼을 실행 시킵니다.

```
void __fastcall THostAppForm::FormCreate(TObject *Sender)
{
    startTesting();              // 폼생성시 startTesting() 호출
}

void __fastcall THostAppForm::startTesting()
{
    ITestRunner runner;          //Test 관리 App 객체 선언

    runner.addTest( MulticasterTest::suite() ); // suite 추가
    runner.addTest( ExampleTestCase::suite() ); // suite 추가

    runner.run ();               //Test 관리 App 실행
  }
```

왜 이렇게 번거롭게 만들어 졌을까라는 생각을 해봤습니다. 그냥 테스트 폼 실행 시켜도 되지 않을까 하고 말입니다. 나중에 적겠지만, BCB 포트이기 때문에 이런 구조가 생긴 것 같습니다. 물론 다른 포트가 이런지는 다 확인을 못해봤습니다만, 최소한 툴에 포함되어 있는 포트들은 테스트 폼만 사용되어 집니다.

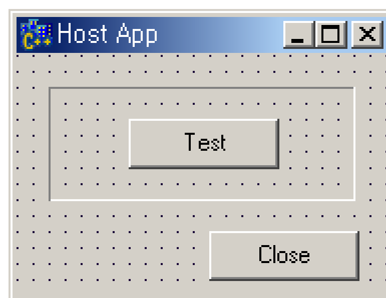


Fig. 5 기본 호스트 폼

이렇게 되어 있는 호스트 폼을 이렇게 수정하면 어떨까요?

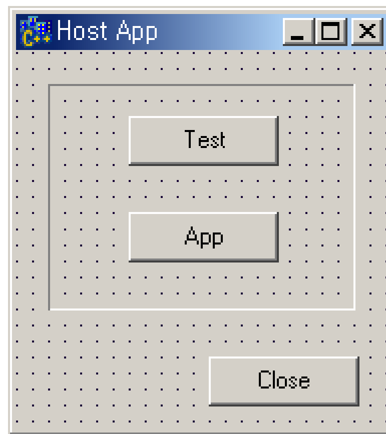


Fig. 6 수정된 호스트 폼

눈치 빠른 분들은 이미 알아 차렸겠죠. 테스트와 릴리즈 용을 별도로 운영할 수 가 있게 되는 겁니다. 테스트를 하고 릴리즈할 프로젝트로 다시 통합하고 빌드하는 일이 반복되겠지요. 사실 TDD(Test Driven Development)가 XP(eXtreme Programming)에 포함된 방법이기 때문에 배포 문제는 TDD의 역할이 아닙니다. XP에서 제안하는 방법론 중에 잦은 릴리즈 라는 항목이 있습니다. 최대한 자주 배포본을 만들어라는 것이지요. 이유는 XP 사이트에서 참조하시고(http://xper.org), 어쨌든 자주 통합 코드를 만들어야 한다는 얘기가 되는데요 BCB에서는 수동으로 하는 수밖에 없는 것 같습니다. 이 부분에 대해서는 추후에 다시 다루게 될 것 입니다. 참고로 JBuilder나 Eclips에서는 Ant(http://ant.apache.org)를 이러한 목적에 사용합니다. Ant는 make 같은 build 툴인데요, 종속성을 없애기 위한 목적으로 개발되었습니다. XML 양식과 shell 스타일이 합쳐진 것입니다.

* ITestRunner.cpp, h

TestRunnerProject.dll에 포함되어 있는 Test 목적의 클래스입니다. 보여지는 Test Form과 Test에 관계된 클래스를 포함합니다. 위에서 제시했던 UnitTest를 위한 4가지의 클래스 뿐만 아니라, UI 및 기타 클래스와 연관 있습니다.

테스트를 어떻게 추가하고, 실제 테스트는 어떻게 하는지를 살펴 봅시다. TDD에서는 모든 것을 의심합니다. x = 1+1 에서 결과로 x가 2인지 조차도 의심을 합니다. 그리고 통과한 테스트 케이스는 누적해서 계속 테스트 합니다. 프로젝트에서 모든 것은 서로 연관이 있기 때문이지요. TDD를 어떻게 적용하는지는 논제에서 좀 벗어납니다. 추후에 실제 프로젝트에 적용하기에서 맛은 볼 수 있겠지만 충분하지는 않을 겁니다.

테스트 케이스를 만들기 위해서는 클래스 작성시 TestCase의 파생클래스로 선언 합니다.

```
class MulticasterTest : public TestCase
{
 …
}
```

그리고 테스트하고 싶으면 method 타입으로 선언하되, 선언 시 test_ 라는 접두어를 사용하면 자동으로 Test Case로 인식하게 됩니다. 즉, **test**SinglePublish() 이렇게 선언하면 된다는 말입니다. 테스트를 suite로 묶기 위해서 suite()로 method를 선언해서 아래와 같이 해주면 suite로 등록이 됩니다.

```
Test *MulticasterTest::suite ()
{
    TestSuite *suite = new TestSuite ("Multicaster");

    suite->addTest (new MulticasterTest("testSinglePublish"));
    suite->addTest (new MulticasterTest
                        ("testMultipleHomogenousPublish"));
    suite->addTest (new MulticasterTest
                        ("testMultipleHeterogenousPublish"));
    suite->addTest (new MulticasterTest("testSingleUnsubscribe"));
    suite->addTest (new MulticasterTest("testMultipleUnsubscribe"));
    suite->addTest (new MulticasterTest("testSimpleUnsubscribeAll"));
    suite->addTest (new MulticasterTest("testComplexUnsubscribeAll"));

    return suite;
}
```

정리하면, Test Case는 TestCase 클래스에서 파생된 클래스의 method를 의미합니다. Suite는 이러한 method의 그룹입니다. 실제 사용은 다음과 같이 합니다.

```
void MulticasterTest::testSinglePublish ()
{
    // Make sure we can subscribe and publish to an address
    Value value;

    assert (m_multicaster->subscribe (m_o1, "alpha"));
    assert (m_multicaster->publish (NULL, "alpha", value));
```

```
        assert (*m_o1 == Observer ("alpha", 1));
    }
```

위의 예제를 보면 단순히 assert(…)를 사용하고 있습니다. 여기가 UnitTest의 가장 하단부라고 생각하시면 됩니다. 모든 테스트는 assert, 즉 확인하는 것으로 진행합니다. 이러한 유형은 몇 가지가 있습니다. BCB 포트에서는 지원하는 것이 한정되어 있지만, 사용하기에는 충분합니다.

assertImplementation () :

assert를 메크로 처리하여 사용되어집니다. 인자로 들어오는 값의 boolean으로 결정합니다.

assertEquals () :

인자로 들어오는 값이 동일한지 비교하여 boolean으로 결정합니다.

이렇게 확인한 결과가 모두 이상이 없으면 다음 Test case를 진행하고 모두 이상이 없으면 Progress Bar가 초록색으로 표시됩니다. 하나라도 이상이 있다면, 빨간색으로 표시되면서 메시지가 표시되겠지요. 컴파일 에러만으로 프로젝트를 진행하기에는 힘든 경우가 많습니다. 언제나 예기치 못한 상황은 생기기 마련이고, 모든 것을 표기하기도 불가능하니까요. TDD 방법을 사용하면 그나마 자신이 만든 코드를 누적 시켜가도 불안한 마음은 사라질 것입니다. UnitTest FrameWork에서 모든 걸 알려줄 테니까요.

이제, 실전에 적용해 봅시다.

< 다음 편에 적겠습니다.>

/**** 이 문서의 저작권은 볼랜드포럼에 있습니다.

***by 김성진.kark*** kark@borlandforum.com ****/