Aeryn

C++ Test Framework
http://www.paulgrenyer.dyndns.org/aeryn

What is Aeryn?

- C++ Testing Framework
- Lightweight
- Adaptable

How does Aeryn Work?

- Test Conditions
- Test Fixtures
- Test Cases
- Running Test Cases
- Reporting

Test Conditions

A test condition is a piece of code being tested using one of Aeryn's test condition macros such as IS_EQUAL.

For example:

```
IS_EQUAL( lifeTheUniverseAndEverything, 42 );
```

Test Conditions

- Macros
- All tests should pass
- Exceptions
  - Failure message
  - Line Number
  - File name

`IS_TRUE( code )`

- Header file: `istrue.h`
- Failure condition: `code` evaluates to false
- Failure message: `IS_TRUE( … )`
- Example:

```
IS_TRUE( lifeTheUniverseAndEverything == 42 );
```

IS_FALSE(code)

- Header file: `istrue.h`
- Failure condition: `code` evaluates to true
- Failure message: `IS_FALSE( … )`
- Example:

```
IS_FALSE( lifeTheUniverseAndEverything == 42 );
```

FAILED(msg)

- Header file: `failed.h`
- Failure condition: Always fails
- Failure message: `msg`

FAILED(msg)

Example:

```
try
{
    …
}
catch( const std::exception& e )
{
    FAILED( e.what() );
}
```

IS_EQUAL(lhs, rhs)

- Header file: `isequal.h`
- Failure condition: `lhs != rhs`
- Failure message:

If the types of `lhs` and `rhs` are streamable a message detailing the failure is created. Eg:

```
Expected '4' but got '2'.
```

IS_EQUAL(lhs, rhs)

If the types are not streamable the test condition is used. Eg:

```
IS_EQUAL( 4, 2 );
```

- Example:

```
IS_EQUAL( lifeTheUniverseAndEverything,
          42 );
```

IS_NOT_EQUAL(lhs, rhs)

- Header file: `isequal.h`
- Failure condition: `lhs == rhs`
- Failure message:

If the types of `lhs` and `rhs` are streamable a message detailing the failure is created. Eg:

```
Expected not to get '42'.
```

IS_NOT_EQUAL(lhs, rhs)

If the types are not streamable the test condition is used. Eg:

```
IS_NOT_EQUAL( 4, 4 );
```

- Example:

```
IS_NOT_EQUAL( lifeTheUniverseAndEverything,
              43 );
```

Test Fixture

A test fixture is a class or a function containing test conditions. For example:

```
void TestForTheMeaningOfLife()
{
    IS_EQUAL( lifeTheUniverseAndEverything,
              42 );
}
```

Function Based Test Fixture

```
void TestForTheMeaningOfLife()
{
    …
    IS_EQUAL( universeAndEverything,
              42 );
}
```

Function Based Test Fixture

```
void TestForTheMeaningOfLife
    ( int universeAndEverything )
{
    IS_EQUAL( universeAndEverything,
              42 );
}
```

Class Based Test Fixture

```
class CalculatorTest
{
private:
    Calculator* calc_;
public:
    CalculatorTest()
        { calc_ = new Calculator; }

    ~CalculatorTest()
        { delete calc_; }

    void TestBasics()
    {   double result = calc_->evaluate("1 + 1");
        IS_TRUE(2.0 == result); }
};
```

Class Based Test Fixture

```
class TestForTheMeaningOfLife
{
private:
    const int universeAndEverything_;
public:
    TestForTheMeaningOfLife
     ( int universeAndEverything )
      : universeAndEverything_
             ( universeAndEverything )
    {
    }
    …
};
```

Test Case

A test case is a wrapper for a test fixture which enables it to be given a name and run by Aeryn.

Test Case: Function Based Test Fixture

```
TestCase( “Universe and everything",
       TestForUniverseAndEverything);
```

or

```
TestCase(
  TestForUniverseAndEverything );
```

Test Case: Function Based Test Fixture

```
const int universeAndEverything = 42;
…
TestCase( “Universe and everything",
   FunctionPtr(
    TestForUniverseAndEverything,
    lifeTheUniverseAndEverything ) );
```

Test Case: Function Based Test Fixture

```
const int universeAndEverything = 42;
…
TestCase(
  FunctionPtr(
    TestForUniverseAndEverything,
    lifeTheUniverseAndEverything ) );
```

Test Case: Class Based Test Fixture

```
TestCase( "Basics",
  Incarnate(
   &CalculatorTest::TestBasics )
  );
```

or

```
TestCase( Incarnate(
  &CalculatorTest::Run ) );
```

Test Case: Class Based Test Fixture

```
const int universeAndEverything = 42;

…

TestCase( “Universe and everything",
 Incarnate(
  &TestForUniverseAndEverything::Run,
            universeAndEverything ) );
```

Test Case: Class Based Test Fixture

```
const int universeAndEverything = 42;

…

TestCase( Incarnate(
  &TestUniverseAndEverything::Run,
           universeAndEverything ) );
```

USE_NAME Macro

```
TestCase( USE_NAME(
  TestForUniverseAndEverything ) );

TestCase( USE_NAME( FunctionPtr(
         TestForUniverseAndEverything,
          lifeTheUniverseAndEverything ))));
```

Test For Universe And everything

Adding and Running Test Cases

```
#include <aeryn/testrunner.h>
…

using namespace Aeryn;
TestRunner testRunner;
```

Adding a Single Test Case

```
testRunner.Add
(
   "HHGTTG",
   TestCase( USE_NAME(
    TestForTheMeaningOfLife ) )
);
```

Adding a Single Test Case

```
testRunner.Add
(
   TestCase(
    USE_NAME(
      TestForTheMeaningOfLife ))
);
```

Adding an Array of Test Cases

```
TestCase calculatorTests[] =
{
    TestCase( "Basics",
      Incarnate( &CalculatorTest::TestBasics ) ),
    TestCase( "Variables",
      Incarnate( &CalculatorTest::TestVariables ) ),
    TestCase( "Compound",
      Incarnate( &CalculatorTest::TestCompound ) ),
    TestCase()
};

...

testRunner.Add( "Calculator", calculatorTests );
```

Adding an Array of Test Cases

```
TestCase calculatorTests[] =
{
    TestCase( "Basics",
      Incarnate( &CalculatorTest::TestBasics ) ),
    TestCase( "Variables",
      Incarnate( &CalculatorTest::TestVariables ) ),
    TestCase( "Compound",
      Incarnate( &CalculatorTest::TestCompound ) ),
    TestCase()
};

...

testRunner.Add( calculatorTests );
```

USE_NAME Macro

```
testRunner.Add(
  USE_NAME( calculatorTests ) );
```

Calculator Tests

Running Tests

```
testRunner.Run();
```

Reports

- Minimal
- Verbose
- GCC
- XCode
- Custom

Minimal Report

```
Aeryn 2 (c) Paul Grenyer 2005
http://www.paulgrenyer.dyndns.org/aeryn
-----------------------------------------

Ran 23 tests, 23 Passed, 0 Failed.
```

Minimal Report

```
Aeryn 2 (c) Paul Grenyer 2005
http://www.paulgrenyer.dyndns.org/aeryn
-----------------------------------------

Test    : IsTrue( true )
Failure : IS_TRUE( true ) failed.
Line    : 48
File    : c:\sandbox\...\testfunctest.cpp
-----------------------------------------

Ran 23 tests, 22 Passed, 1 Failed.
```

Verbose Report

```
Aeryn 2 (c) Paul Grenyer 2005
http://www.paulgrenyer.dyndns.org/aeryn
-----------------------------------------
Test Set : Report Tests

 - Minimal Report Test
 - Verbose Report Test
 - Gcc Report Test
-----------------------------------------
...

Ran 23 tests, 23 Passed, 0 Failed.
```

Verbose Report

```
...

Test Set : Test Function Tests

 - IsTrue( true )

Failure : IS_TRUE( true ) failed.
Line    : 48
File    : c:\sandbox\...\testfunctest.cpp
...
-----------------------------------------
Ran 23 tests, 22 Passed, 1 Failed.
```

GCC Report

```
Aeryn 2 (c) Paul Grenyer 2005
http://www.paulgrenyer.co.uk/aeryn

testfunctest.cpp:48: error: Test Failure -
  'IsTrue( true )' in 'Test Function Tests'
  - IS_TRUE( true ) failed.
```

XCode Report

- The XCode [XCode] report is the same as the GCC report, but uses a file based to cookie to stop the tests begin run if the code has not been changed.

Controlling Reports with Command Line Arguments

```
int main( int argc, char *argv[] )
{
    using namespace Aeryn;

    TestRunner testRunner;
    Aeryn::AddTests( testRunner );

    TestRunner::IReportPtr
              report(
    TestRunner::CreateReport( argc, argv) );

    return testRunner.Run( *report.get() );
}
```

Controlling Reports with Command Line Arguments

-
- `verbose`
- `gcc`
- `xcode`

Custom Reports

- Implement IReport interface.
  - `BeginTesting`
  - `BeginTestSet`
  - `BeginTest`
  - `Pass`
  - `Failure`
  - `Error`
  - …

Interfaces and Mock Objects

- Tests should take ZERO time to run.
  - What about file, database, and network access tests?
- Mock objects
- The need for interfaces

Mock Object

A mock object is an object that can be used in place of another to mimic its expected behaviour.

Interfaces

Interfaces are needed to make the real and mock objects easily interchangeable.

File Interface

```
class IFile
{
protected:
    IFile(){}

public:
    virtual ~IFile() = 0 {}

    virtual bool Open
      ( const std::string& filename ) = 0;

    virtual void Close() = 0;

    virtual std::string Read() = 0;
};
```

Real File Object

```
class File : public IFile
{
private:
    // ...

public:
    bool Open( const std::string& filename )
    {  // Real code which opens a file. }

    void Close()
    {  // Real code which closes the file. }

    std::string Read()
    {  // Real code which reads a line from th file. }
};
```

Mock File Object (1)

```
class MockFile : public IFile
{
public:
    typedef std::vector< std::string > LineCont;

    MockFile( const LineCont lines )
        : line_( 0 ), lines_( lines )
    {
    }

    ...

private:
    int line_;
    const LineCont lines_;
};
```

Mock File Object (2)

```
class MockFile : public IFile
{
public:
   ...

    bool Open( const std::string& filename )
    {   line_ = 0;
        return true; }

    void Close()
    {}

    std::string Read()
    {   std::string result = lines_[ line_ ];
        ++line_;
        return result; }
    ...
};
```

File Processor

```
class FileProcessor
{
private:
    IFile& file_;

public:
    FileProcessor( IFile& file )
      : file_( file )
    {
    }

     …
};
```

Production Code

```
File file;
FileProcessor( file );
…
```

Test Code

```
MockFile::LineCont lines;
lines.push_back( “…” );
lines.push_back( “…” );
…

MockFile mockFile( lines );
FileProcessor processor( lines );
…
```

Round Up

- How to create and run Aeryn tests.
- How to use and create Aeryn reports.
- Interfaces and mock objects.
- Any (more) questions?