Contents

1 概要

这是一个Win32的单元测试框架，主要解决如下问题：
1. 测试用例的依赖关系：由于函数、方法对象之间存在依赖关系， 测试用例自然也应该存在依赖关系，因此需要有一种方法来表达这种依赖关系。
2. 运行特定的测试用例：在TDD开发中，这是很常见的情况。
3. 快速定位测试失败位置：当某个测试用例失败时， 能够快速地定位到源文件以及行数是可以很大地提高开发效率的。
4. 提供一定格式的日志功能：以便日志工具进行过滤和归类
5. 提供日志输出的配置功能：以便在开发时忽略不必要的信息

整个测试框架基于Windows API， 依赖boost.program_options, boost.unordered, boost.filesystem, boost.regex库， C++标准使用C++2003， 构建和工程管理使用cmake。

2 用法

基本的测试用例写法见第9节。

2.1 编写测试程序

2.2 编写一个单元测试

2.3 定义测试断言

2.4 使用测试日志

2.5 运行测试程序

2.5.1 运行和输出

2.5.2 运行特定测试用例

2.5.3 调整日志信息

3 设计

测试框架以树形结构组织测试用例，因此在框架中有TestTree类， 由于在测试程序的一次运行中，需要运行的测试用例是固定不变的， 故测试用例树也是不变的，因此TestTree提供了单例模式。

由于需要通过参数对测试程序的运行做出一些调整。 如运行特定的测试用例或者调整日志信息等。 因此测试框架提供了TestOptions这个类来管理这些参数。 由于在测试过程中，这些参数也是不变的，故TestOptions也是单例(Singleton)。

测试框架提供了一系列的日志工具和断言工具， 以便用户方便地管理测试用例和测试信息。

最后，测试框架提供了一个测试程序的主入口(WinMain函数)的定义。 如果用户不想自定义测试程序的主入口，则可以直接使用测试框架提供的主入口。

测试框架以源文件而非库的方式提供，以便保证在不同的编译器下的最大兼容性。

4 CMakeLists.txt

这是整个测试框架的工程管理文件，虽然测试框架以源文件的方式提供， 但测试框架本身也需要一些单元测试以及示例程序，因此需要这个工程管理文件。

⟨tangle⟩≡
```
notangle -RCMakeLists.txt win32gui.nw>CMakeLists.txt
```

⟨CMakeLists.txt⟩≡
```
project(win32gui)
include_directories("${PROJECT_SOURCE_DIR}/boost_1_54_0")
link_directories("${PROJECT_SOURCE_DIR}/boost_1_54_0/lib")
add_library(testframework ⟨sources of tesframework⟩)
```

5 test_main.hxx

test_main.hxx这个文件包含了一个测试程序的主入口， 如果用户对测试程序的主入口没有什么特殊要求的话， 则应该使用这个文件中的入口作为测试程序的主入口。

使用测试框架提供的主入口很简单，只要在测试程序的任意一个源文件中添加如下一行即可。

```
WIN32GUITESTMAIN
```

⟨tangle⟩+≡

```
if [ ! -d src ]
then
    mkdir src
fi
notangle -R"test\\_main.hxx" -t4 -L'#line %L "%F"%N' win32gui.nw>src/test_main.hxx
includeDir="include"
if [ ! -d $includeDir ]
then
    mkdir $includeDir
fi
cp src/test_main.hxx $includeDir
```

Windows的程序以WinMain为主入口，在WinMain函数中，测试框架注册和建立了一个窗口，并指定了新窗口的消息处理函数。在消息处理函数中，当接收到窗口已经处于显示状态的消息后，测试框架就依次运行测试用例，然后退出。

⟨test_main.hxx⟩≡

```
#include <windows.h>
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include <tchar.h>
#include <iostream>
#define WIN32GUITESTMAIN  \
extern "C"{ \
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM); \
const char procName[] = "win32guitest"; \
HINSTANCE g_appInstance = NULL; \
 \
int WINAPI WinMain(HINSTANCE inst, HINSTANCE prevInst, LPSTR lpszCmdLine, int nCmdShow) \
{ \
    using boost::program_options::split_winmain; \
    using OpenGUIL::TestFramework::TestOptions; \
    using std::vector; \
    using std::string; \
    vector<string> args = split_winmain(lpszCmdLine); \
    TestOptions::instance()->parseOptions(args); \
    HWND hWnd = NULL; \
    g_appInstance = inst; \
    MSG lpMsg; \
    WNDCLASS wcApp; \
    memset(&wcApp, 0, sizeof(wcApp)); \
     \
    wcApp.lpszClassName = procName; \
    wcApp.hInstance = inst; \
    wcApp.lpfnWndProc = WndProc; \
    wcApp.hCursor = LoadCursor(NULL, IDC_ARROW); \
    wcApp.hIcon = NULL; \
    wcApp.lpszMenuName = 0; \
    wcApp.hbrBackground = (HBRUSH) GetStockObject (BLACK_BRUSH); \
    wcApp.style = CS_HREDRAW | CS_VREDRAW; \
    wcApp.cbClsExtra = NULL; \
    wcApp.cbWndExtra = NULL; \
    if (!RegisterClass(&wcApp)) \
    { \
        return 0; \
    } \
 \
    hWnd = CreateWindow(procName, \
                        "win32guitest", \
                        WS_OVERLAPPEDWINDOW, \
                        CW_USEDEFAULT, \
                        CW_USEDEFAULT, \
                        CW_USEDEFAULT, \
                        CW_USEDEFAULT, \
                        (HWND)NULL, \
                        (HMENU)NULL, \
                        inst, \
                        (LPSTR)NULL); \
    ShowWindow(hWnd, nCmdShow); \
    UpdateWindow(hWnd); \
    while (GetMessage(&lpMsg, 0, 0, 0)) \
    { \
        TranslateMessage(&lpMsg); \
        DispatchMessage(&lpMsg); \
    } \
    return lpMsg.wParam; \
} \
 \
static int retCode = 0; \
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) \
{ \
    using OpenGUIL::TestFramework::TestTree; \
    using OpenGUIL::TestFramework::TestOptions; \
    using std::cout; \
    using std::endl; \
    static bool testcasesRunned = false; \
    switch (message) \
    { \
        case WM_ACTIVATE: \
            { \
                if (true == testcasesRunned) \
                { \
                    break; \
                } \
                testcasesRunned = true; \
                TEST_LOG("start testing"); \
                OpenGUIL::TestFramework::setupWIN32TestEnv(g_appInstance, hWnd); \
                TestOptions::TestcaseNames testcaseNames = \
                    TestOptions::instance()->specifiedTestcases(); \
                if (0 == testcaseNames.size()) \
                { \
                    testcaseNames.push_back(""); \
                } \
                for (TestOptions::TestcaseNames::iterator iter = \
                        testcaseNames.begin(); \
                     iter != testcaseNames.end(); \
                     iter++) \
                { \
                    if (false == TestTree::instance()->runTestCases(*iter)) \
                    { \
                        retCode = -1; \
                    } \
                } \
                if (!TestOptions::instance()->shouldConfirmQuitting()  \
                || true == userConfirm("All test case run, quit?", hWnd)) \
                { \
                    PostQuitMessage(retCode); \
                } \
            } \
            break; \
        case WM_DESTROY: \
            PostQuitMessage(retCode); \
            break; \
        default: \
            return DefWindowProc(hWnd, message, wParam, lParam); \
            break; \
    } \
    return (0); \
} \
}
```

6 test_tools.hpp

这个文件提供了测试框架的一些工具。

⟨tangle⟩+≡

```
chunk=test\\_tools.hpp
target=test_tools.hpp
targetDir=src
targetPath=$targetDir/$target
file=win32gui.nw
if [ ! -d $targetDir ]
then
    mkdir $targetDir
fi
notangle -t4 -L'#line %L "%F"%N' -R"$chunk" $file>$targetPath
includeDir="include"
if [ ! -d $includeDir ]
then
    mkdir $includeDir
fi
cp $targetPath $includeDir
```

⟨test_tools.hpp⟩≡

```
#ifndef TEST_TOOLS_HPP
#define TEST_TOOLS_HPP
⟨Head section of test_tools.hpp⟩
namespace OpenGUIL{
namespace TestFramework{
    ⟨Contents of test_tools.hpp⟩
}
}
#endif
```

6.1 TestFramework::LogPlace

这是一个很简单的结构体，用于记录日志发生的地点，包括文件名、行号和函数签名。 同时提供输出到输出流中的方法。

⟨Contents of test_tools.hpp⟩≡

```
struct LogPlace
{
    const char* _fileName;
    int _line;
    const char* _function;
    LogPlace(const char* fileName, int line, const char* func):
        _fileName(fileName),
        _line(line),
        _function(func)
    {}
};
template <typename StreamT>
StreamT& operator<<(StreamT& strm, const LogPlace& l)
{
    using boost::filesystem::path;
    strm<<path(l._fileName).filename()<<": "<<l._line<<", "<<l._function;
    return strm;
}
```

⟨Head section of test_tools.hpp⟩≡

```
#include <boost/filesystem.hpp>
```

6.2 TestFramework::toString

这个函数模板用于提供类型安全的printf。用法很简单：

```
using namespace std;
stringstream sstrm;
toString(sstrm, "hello world, ", 123, 1.2, time(NULL));
```

这个函数模板的实现则比较繁琐，利用了函数重载机制， 这是因为C++/03不支持变参模板的缘故。

测试框架提供了最多30个参数的形式，如果不够用，用户可以自己实现更多的参数。

⟨Contents of test_tools.hpp⟩+≡

```
/*
template<typename HeadT, typename... RestT>
std::string toString(std::stringstream& strm, const HeadT& head, RestT... rest)
{
    strm<<head;
    return toString(strm, rest...);
}
template<typename HeadT>
std::string toString(std::stringstream& strm, const HeadT& head)
{
    strm<<head;
    return strm.str();
}
*/
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
  return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16, _17, _18
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
  return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16, _17, _18
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
  return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16, _17, _18
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
  return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16, _17, _18
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
  return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16, _17, _18
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
  return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16, _17, _18
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
   strm<<_0;
  return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16, _17, _18
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
   strm<<_0;
  return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16, _17, _18
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
   strm<<_0;
  return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16, _17, _18
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
   strm<<_0;
  return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16, _17, _18
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
   strm<<_0;
  return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16, _17, _18
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
   strm<<_0;
  return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16, _17, _18
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
   strm<<_0;
  return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16, _17 );
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
   strm<<_0;
  return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16 );
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
    return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15 );
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
    return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14 );
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
    return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13 );
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
    return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12 );
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
    return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11 );
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
    return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10 );
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
    return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8, _9 );
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
    return toString (strm, _1, _2, _3, _4, _5, _6, _7, _8 );
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
    return toString (strm, _1, _2, _3, _4, _5, _6, _7 );
```

```
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5, type
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
    return toString (strm,  _1,  _2,  _3,  _4,  _5,  _6 );
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4, typename T5 >
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4, const
{
    strm<<_0;
    return toString (strm,  _1,  _2,  _3,  _4,  _5 );
}
template <typename StreamT, typename T0, typename T1, typename T2, typename T3, typename T4 >
std::string toString (StreamT& strm, const T0& _0, const T1& _1, const T2& _2, const T3& _3, const T4& _4 )
{
    strm<<_0;
    return toString (strm,  _1,  _2,  _3,  _4 );
}
template <typename StreamT,  typename T0,  typename T1,  typename T2,  typename T3 >
std::string toString (StreamT& strm,  const T0& _0,  const T1& _1,  const T2& _2,  const T3& _3 )
{
    strm<<_0;
    return toString (strm,  _1,  _2,  _3 );
}
template <typename StreamT,  typename T0,  typename T1,  typename T2 >
std::string toString (StreamT& strm,  const T0& _0,  const T1& _1,  const T2& _2 )
{
    strm<<_0;
    return toString (strm,  _1,  _2 );
}
template <typename StreamT,  typename T0,  typename T1 >
std::string toString (StreamT& strm,  const T0& _0,  const T1& _1 )
{
    strm<<_0;
    return toString (strm,  _1 );
}
template <typename StreamT,  typename T0 >
std::string toString (StreamT& strm,  const T0& _0 )
{
    strm<<_0;
    return strm.str();
}
```

6.3 TEST_LOG

日志宏。

⟨Head section of test_tools.hpp⟩+≡

```
#include <sstream>
#include <iostream>
#define TEST_LOG(...) \
{ \
    std::stringstream strm; \
    std::stringstream strm2; \
    strm<< OpenGUIL::TestFramework::LogPlace(__FILE__, __LINE__, __FUNCTION__) \
        <<": " \
        << OpenGUIL::TestFramework::toString(strm2, __VA_ARGS__) \
        <<std::endl; \
    printf("%s", strm.str().c_str()); \
}
```

6.4 TEST_SUMMARY

用于输出每个单元测试的概况。

⟨Head section of test_tools.hpp⟩+≡

```
#define TEST_SUMMARY(...)  \
{ \
    std::stringstream strm; \
    std::stringstream strm2; \
    strm<< OpenGUIL::TestFramework::toString(strm2, __VA_ARGS__)<<std::endl; \
    printf("%s", strm.str().c_str()); \
}
```

6.5 TEST_MESSAGE

用户使用的宏，用于输出用户信息。程序采用正则表达式的方式来控制用户信息的输出，当指定正则表达式时，只有符合该正则表达式的用户信息才会被输出。具体的过滤方法参见TestOption。

⟨Head section of test_tools.hpp⟩+≡

```
#define TEST_MESSAGE(...) \
{ \
    std::stringstream strm; \
    using OpenGUIL::TestFramework::TestOptions; \
    OpenGUIL::TestFramework::toString(strm, __VA_ARGS__); \
    std::string str = strm.str(); \
    if (TestOptions::instance()->filterUserMessage(str)) \
    { \
        printf("%s\n", str.c_str()); \
    } \
}
```

6.6 TEST_PASS

TEST_PASS测试断言宏。当断言失败时，这个宏会输出失败的条件以及地点等信息。

⟨Head section of test_tools.hpp⟩+≡
```
#define TEST_PASS(condition)  \
{ \
    std::string cond = #condition; \
    if (condition) \
    { \
        this->result = true; \
    } \
    else \
    { \
     TEST_LOG("test condition \"", cond, "\" is not match, testcase \"", testCaseName(), "\" failed."); \
        this->result = false; \
        return; \
    } \
}
```

7 win32gui_unit_test.hpp

测试框架的主要头文件。

⟨tangle⟩+≡
```
chunk=win32gui\\_unit\\_test.hpp
target=win32gui_unit_test.hpp
targetDir=src
targetPath=$targetDir/$target
file=win32gui.nw
if [ ! -d $targetDir ]
then
    mkdir $targetDir
fi
notangle -t4 -L'#line %L "%F"%N' -R"$chunk" $file>$targetPath
includeDir="include"
if [ ! -d $includeDir ]
then
    mkdir $includeDir
fi
cp $targetPath $includeDir
```

```
⟨win32gui_unit_test.hpp⟩≡
  #ifndef WIN32GUI_UNIT_TEST_HPP
  #define WIN32GUI_UNIT_TEST_HPP
  #include <boost/program_options/cmdline.hpp>
  #include <boost/program_options/options_description.hpp>
  #include <boost/program_options/variables_map.hpp>
  #include <boost/program_options/parsers.hpp>
  #include <boost/program_options/value_semantic.hpp>
  #include <boost/typeof/typeof.hpp>
  #include <boost/unordered_map.hpp>
  ⟨Head section of win32gui_unit_test.hpp⟩
  ⟨Preamble of win32gui_unit_test.hpp⟩
  namespace OpenGUIL{
  namespace TestFramework{
  ⟨Contents of win32gui_unit_test.hpp⟩
  }
  }
  #include "test_main.hxx"
  #endif
```

8 win32gui_unit_test.cpp

测试框架的实现文件。

```
⟨tangle⟩+≡
  chunk=win32gui\\_unit\\_test.cpp
  target=win32gui_unit_test.cpp
  targetDir=src
  targetPath=$targetDir/$target
  file=win32gui.nw
  if [ ! -d $targetDir ]
  then
      mkdir $targetDir
  fi
  notangle -t4 -L'#line %L "%F"%N' -R"$chunk" $file>$targetPath
```

```
⟨sources of tesframework⟩≡
  src/win32gui_unit_test.cpp
```

```
⟨win32gui_unit_test.cpp⟩≡
  ⟨Head section of win32gui_unit_test.cpp⟩
  #include "win32gui_unit_test.hpp"
  namespace OpenGUIL{
  namespace TestFramework{
  ⟨Contents of win32gui_unit_test.cpp⟩
  }
  }
```

8.1 TestFramework::TestOptions

`TestOptions`用于管理单元测试中的各种配置（一般通过参数输入）。这个类在程序的整个生命周期中只有一个实例，故提供了访问这个实例的单例方法`instance()`。

⟨Contents of win32gui_unit_test.hpp⟩≡

```
struct TestOptions
{
    ⟨Contents of struct TestOptions⟩
};
```

8.1.1 TestOptions::instance

获取单例，非线程安全。

⟨Contents of struct TestOptions⟩≡

```
public:
static TestOptions* instance()
{
    static TestOptions* instance = NULL;
    if (instance == NULL)
    {
        instance = new TestOptions();
    }
    return instance;
}
```

8.1.2 TestOptions::parseOptions

这个成员函数用于解析测试程序得到的命令行参数。

⟨Contents of struct TestOptions⟩+≡

```
private:
boost::program_options::variables_map variableMap_;
boost::program_options::options_description optionsDescription_;
public:
template <typename RawOptions>
void parseOptions(const RawOptions& rawOptions)
{
    using boost::program_options::store;
    using boost::program_options::notify;
    using boost::program_options::command_line_parser;

    store(command_line_parser(rawOptions).options(optionsDescription_).run(), variableMap_);
    notify(variableMap_);
}
```

8.1.3 TestOptions::trace

这是测试程序的一个开关，当打开时，测试程序会输出测试框架内部的一些日志信息。 通常用于调试测试框架。

⟨Contents of struct TestOptions⟩+≡
```
bool trace()
{
    return variableMap_.count("trace") > 0;
}
```

8.1.4 TFTRACE

宏 `TFTRACE` 意为 "Test Framework Trace", 用于对TestFramework自身的debug。

⟨Preamble of win32gui_unit_test.hpp⟩≡
```
#define TFTRACE(...)  \
{ \
    if (TestOptions::instance()->trace()) \
    { \
        TEST_LOG(__VA_ARGS__); \
    } \
}
```

8.1.5 TestOptions::shouldConfirmQuitting

这是测试程序的一个开关，当打开时，测试程序在测试完成后，即将退出前需要用户确认。

⟨Contents of struct TestOptions⟩+≡
```
bool shouldConfirmQuitting()
{
    return variableMap_.count("should-confirm-quit") > 0;
}
```

8.1.6 TestOptions::TestOptions

`TestOptions`的构造函数。

⟨Contents of struct TestOptions⟩+≡
```
TestOptions():optionsDescription_("Unit test options")
{
    using boost::program_options::value;
    optionsDescription_.add_options()
        ("trace", "print trace message in test framework")
        ("should-confirm-quit", "user confirm quiting after all test case run")
     ("filter-user-message", value<std::string>(), "use regex to filter user messages, only matched can be display
        ("testcases", value<std::string>(), "running specific testcases, "\
        "use comma to seperate testcase names");
}
```

8.1.7 TestOptions::filterUserMessage

这个选项用于过滤单元测试中用户用TestMessage输出的用户信息。参数的值是正则表达式。当某条用户信息与这条正则表达式匹配时，该条用户信息将被输出。当不指定正则表达式时，输出所有的用户信息。当正则表达式的模式错误时，也输出所有的用户信息。

⟨Head section of test_tools.hpp⟩+≡

```
#include <boost/regex.hpp>
```

⟨Contents of struct TestOptions⟩+≡

```
private:
boost::regex userMessageFilter_;
public:
bool filterUserMessage(const std::string& message)
{
    if (variableMap_.count("filter-user-message") == 0)
    {
        TFTRACE("no filter pattern for user message");
        return true;
    }
    if (userMessageFilter_.empty())
    {
        std::string pattern =
            variableMap_["filter-user-message"].as<std::string>();
        userMessageFilter_.assign(pattern, boost::regex_constants::match_any);
    }
    if (0 != userMessageFilter_.status())
    {
        TFTRACE("ill legal pattern for user message");
        return true;
    }
    TFTRACE("will test message \"", message, "\" for pattern ", userMessageFilter_);
    return regex_search(message, userMessageFilter_);
}
```

8.1.8 TestOptions::specifiedTestcases

用户指定要运行的测试用例，当没有指定任何测试用例，即返回空集合时，表示运行所有测试用例。

⟨Head section of test_tools.hpp⟩+≡

```
#include <boost/algorithm/string.hpp>
```

⟨Contents of struct TestOptions⟩+≡

```
public:
typedef std::vector<std::string> TestcaseNames;
private:
TestcaseNames specifiedTestcases_;
public:
TestcaseNames specifiedTestcases(void)
{
    if (variableMap_.count("testcases") == 0)
    {
        return specifiedTestcases_;
    }
    if (specifiedTestcases_.size() > 0)
    {
        return specifiedTestcases_;
    }
    std::string testcaseNames =
        variableMap_["testcases"].as<std::string>();
    using namespace boost::algorithm;
    split (specifiedTestcases_,
           testcaseNames,
           is_any_of(std::string(", ")),
           token_compress_on);
    return specifiedTestcases_;
}
```

8.2 TestFramework::WIN32TestEnv

对于一个Windows GUI的测试框架，对每个测试用例需要提供一些基本的资源，比如父窗口、程序实例等。WIN32TestEnv这个类的用途就是管理这些基本的资源。同样，这也是个单例。

⟨Contents of win32gui_unit_test.hpp⟩+≡

```
struct WIN32TestEnv
{
private:
     HINSTANCE _hInst;
     HWND _hWnd;
     boost::function<HWND (void)> newWndCreator;
     boost::function<void (HWND)> wndDestroyer;
public:
     void setNewWndCreator(const boost::function<HWND (void)>& newCreator);
     void setWndDestroyer(const boost::function<void (HWND)>& newDestroyer);
     HINSTANCE hInstance();
     void hInstance(HINSTANCE newInstance);
     HWND hWnd();
     void hWnd(HWND newWnd);
     void cleanup();
     static WIN32TestEnv* instance();
};
```

⟨Contents of win32gui_unit_test.cpp⟩≡

```
WIN32TestEnv* WIN32TestEnv::instance()
{
    static WIN32TestEnv* testEnv = NULL;
    if (testEnv == NULL)
    {
        testEnv = new WIN32TestEnv();
        testEnv->hInstance(NULL);
        testEnv->hWnd(NULL);
    }
    return testEnv;
}
void WIN32TestEnv::setNewWndCreator(const boost::function<HWND (void)> & newCreator)
{
    newWndCreator = newCreator;
}
void WIN32TestEnv::setWndDestroyer(const boost::function<void (HWND)> & newDestroyer)
{
    wndDestroyer = newDestroyer;
}
HINSTANCE WIN32TestEnv::hInstance()
{
    return ::GetModuleHandle(NULL);
}
 void WIN32TestEnv::hInstance(HINSTANCE newInstance)
 {
    _hInst = newInstance;
 }
HWND WIN32TestEnv::hWnd()
{
    if( _hWnd == NULL && false == newWndCreator.empty())
    {
        hWnd(newWndCreator());
    }
    return _hWnd;
}
void WIN32TestEnv::hWnd(HWND newWnd)
{
    _hWnd = newWnd;
}
void WIN32TestEnv::cleanup()
{
    if (false == wndDestroyer.empty())
    {
        wndDestroyer(hWnd());
        hWnd(NULL);
    }
}
```

8.3 TestFramework::TestCase

⟨Contents of win32gui_unit_test.hpp⟩+≡

```
struct TestCase
{
    bool result;
    std::string messages;
    TestCase(): result(false){}
    virtual void test() = 0;
    virtual void runTest() = 0;
    virtual const char* testCaseName() const = 0;
    virtual ~TestCase(){};
};
```

8.4 TestFramework::WIN32guiTestCase

⟨Head section of win32gui_unit_test.hpp⟩≡

```
#include <windows.h>
#include "test_tools.hpp"
```

⟨Contents of win32gui_unit_test.hpp⟩+≡

```
struct WIN32GUITestCase: TestCase
{
    HINSTANCE hInstance;
    HWND hWnd;
    virtual void runTest()
    {
        hInstance = WIN32TestEnv::instance()->hInstance();
        hWnd = WIN32TestEnv::instance()->hWnd();
        test();
        WIN32TestEnv::instance()->cleanup();
        hInstance = NULL;
        hWnd = NULL;
    }
};
```

8.5 TestFramework::setupWIN32TestEnv(HINSTANCE hInst, HWND parentWnd)

⟨Contents of win32gui_unit_test.hpp⟩+≡

```
void setupWIN32TestEnv(HINSTANCE hInst, HWND parentWnd);
```

⟨Contents of win32gui_unit_test.cpp⟩+≡

```
void setupWIN32TestEnv(HINSTANCE hInst, HWND parentWnd)
{
    //WIN32TestEnv::instance()->hInstance(hInst);
    WIN32TestEnv::instance()->hWnd(NULL);
    ⟨register test window class⟩
    ⟨setup test window creator and destroyer⟩
}
```

⟨register test window class⟩≡

```
struct Local
{
    static LRESULT WINAPI TestWndMsgProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
    {
        switch (msg)
        {
            case WM_TIMER:
                TFTRACE("a timer message, will destroy this test window(", hWnd, ")");
                DestroyWindow(hWnd);
                return 0;
                break;
            case WM_DESTROY:
                TFTRACE("window ", hWnd, " being destroyed");
                return 0;
                break;
            default:
                return DefWindowProc(hWnd, msg, wParam, lParam);
        };
    }
};
WNDCLASS wcTest;
memset(&wcTest, 0, sizeof(wcTest));
static const char testWndClsName[] = "OpenGUILTestWindow";
wcTest.lpszClassName = testWndClsName;
wcTest.hInstance = hInst;
//wcTest.lpfnWndProc = &DefWindowProc;
wcTest.lpfnWndProc = &Local::TestWndMsgProc;
wcTest.hCursor = LoadCursor(NULL, IDC_ARROW);
wcTest.hIcon = NULL;
wcTest.lpszMenuName = 0;
wcTest.hbrBackground = (HBRUSH) GetStockObject (LTGRAY_BRUSH);
wcTest.style = CS_HREDRAW | CS_VREDRAW;
wcTest.cbClsExtra = NULL;
wcTest.cbWndExtra = NULL;
if (!RegisterClass(&wcTest))
{
    return;
}
```

⟨setup test window creator and destroyer⟩≡

```
struct NewCreator
{
    HINSTANCE hInst;
    HWND parentWnd;
    NewCreator(HINSTANCE inst, HWND pWnd):
        hInst(inst),
        parentWnd(pWnd)
    {}
    HWND operator() (void)
    {
        TFTRACE("create a test window");
        RECT rect;
        ::GetClientRect(parentWnd, &rect);
        HWND hWnd = CreateWindow ("OpenGUILTestWindow",//WindowClass
                                "TestWindow",//WindowTitle
                                WS_CHILD | WS_VISIBLE,//style
                                CW_USEDEFAULT,//x
                                CW_USEDEFAULT,//y
                                rect.right - rect.left,//width
                                rect.bottom - rect.top,//height
                                (HWND) parentWnd,//parent
                                (HMENU) NULL,//menu
                                hInst,//instance of module
                                (LPSTR) NULL);
        ::ShowWindow(hWnd, SW_SHOW);

        return hWnd;
    }
};
WIN32TestEnv::instance()->setNewWndCreator(boost::function<HWND (void)>(NewCreator(hInst, parentWnd)));

struct NewDestroyer
{
    void operator() (HWND hWnd)
    {
        TFTRACE("a test done, destroying window");
        ::DestroyWindow(hWnd);
    }
};

WIN32TestEnv::instance()->setWndDestroyer(NewDestroyer());
```

8.6 TestFramework::TestTree

⟨Contents of win32gui_unit_test.hpp⟩+≡

```
struct TestTree
{
    ⟨Contents of TestTree⟩
};
```

8.6.1 TestFramework::TestTree::runTestCases

⟨Contents of TestTree⟩≡

```
public:
bool runTestCases(const std::string& testCaseName = "");
```

⟨Contents of win32gui_unit_test.cpp⟩+≡

```
bool TestTree::runTestCases(const std::string& testCaseName)
{
    bool ret = true;
    TestCaseList testCases = testCaseList(testCaseName);
    for(TestCaseList::iterator iter = testCases.begin();
        iter != testCases.end();
        iter++)
    {
        if (false == (*iter)())
        {
            ret = false;
        }
    }
    return ret;
}
```

8.6.2 TestFramework::TestTree::registerTestCase

⟨Contents of TestTree⟩+≡

```
typedef TestCase*(*TestCaseGenerator)();
struct TestNode
{
    std::vector<TestNode*> _prerequisites;
    TestCaseGenerator _testCase;
    std::string _testCaseName;
    TestNode():_testCase(NULL)
    {
    }
};
typedef boost::unordered_map<std::string, TestNode*> TestNodes;
typedef boost::unordered_set<TestNode*> TestNodeSet;
TestNodes _testNodes;
TestNodeSet _rootNodes;
public:
static TestTree* instance();
void registerTestCase(
    TestCaseGenerator creator,
    const std::string& testCaseName,
    const std::string& prerequisiteTo);
```

⟨Head section of win32gui_unit_test.hpp⟩+≡

```
#include <boost/function.hpp>
#include <boost/unordered_map.hpp>
#include <boost/unordered_set.hpp>
#include <vector>
#include <string>
```

⟨Contents of win32gui_unit_test.cpp⟩+≡

```
TestTree* TestTree::instance()
{
    static TestTree* pInst = new TestTree();
    return pInst;
}

void TestTree::registerTestCase(
    TestCaseGenerator creator,
    const std::string& testCaseName,
    const std::string& prerequisiteTo)
{
    TestNodes::iterator iter = _testNodes.find(testCaseName);
    TestNode* pTestNode = NULL;
    if (iter == _testNodes.end())
    {
        pTestNode = new TestNode();
        pTestNode->_testCaseName = testCaseName;
        _testNodes.insert(std::make_pair(testCaseName, pTestNode));
    }
    else
    {
        if (iter->second->_testCase != NULL)
        {
            //TODO report some error message?
            return;
        }
        pTestNode = iter->second;
    }
    pTestNode->_testCase = creator;

    _rootNodes.insert(pTestNode);
    if (prerequisiteTo == "")
    {
        return;
    }
    else
    {
        TestNodes::iterator followedNodeIter = _testNodes.find(prerequisiteTo);
        if (followedNodeIter == _testNodes.end())
        {
        followedNodeIter = _testNodes.insert(std::make_pair(prerequisiteTo, new TestNode())).first;
            _rootNodes.insert(followedNodeIter->second);
        }
        TestNode* followedTestNode = followedNodeIter->second;
        followedTestNode->_prerequisites.push_back(pTestNode);
        _rootNodes.erase(pTestNode);
    }
}
```

⟨Contents of TestTree⟩+≡

```
public:
void clear()
{
    for (TestNodes::iterator iter = _testNodes.begin();
         iter != _testNodes.end();
         iter++)
    {
        delete iter->second;
    }
    _testNodes.clear();
    _rootNodes.clear();
}
```

⟨source files of utc_win32gui_test⟩≡

```
../src/win32gui_unit_test.cpp
```

⟨Contents of utc_win32gui_test.cpp⟩≡

```
BOOST_AUTO_TEST_CASE(utc_TestFramework_TestTree_registerTestCase)
{
    using OpenGUIL::TestFramework::TestTree;
    TestTree::instance()->registerTestCase(NULL, "rootTestCase", "");
    TestTree::instance()->registerTestCase(NULL, "firstTestCase", "rootTestCase");
    BOOST_CHECK_EQUAL(TestTree::instance()->_testNodes.size(), 2);
    BOOST_CHECK_EQUAL(TestTree::instance()->_rootNodes.size(), 1);
    TestTree::instance()->clear();
    BOOST_CHECK_EQUAL(TestTree::instance()->_testNodes.size(), 0);
    BOOST_CHECK_EQUAL(TestTree::instance()->_rootNodes.size(), 0);
}
```

8.6.3 TestFramework::TestTree::testCaseList

⟨Contents of TestTree⟩+≡

```
public:
typedef std::vector<boost::function<bool(void)> > TestCaseList;
TestCaseList testCaseList(const std::string& testCaseName = "");
```

⟨Contents of win32gui_unit_test.cpp⟩+≡

```
TestTree::TestCaseList TestTree::testCaseList(const std::string& testCaseName)
{
    TestNodeSet* pNodes = NULL;
    TestNodeSet nodeSet;
    TestCaseList ret;
    std::vector<TestNode*> nodes;
    if ( testCaseName == "")
    {
        pNodes = &_rootNodes;
    }
    else
    {
        TestNodes::iterator iter = _testNodes.find(testCaseName);
        if (iter != _testNodes.end())
        {
            nodeSet.insert(iter->second);
        }
        pNodes = &nodeSet;
    }
    if (pNodes != NULL)
    {
        for (TestNodeSet::iterator iter  = pNodes->begin();
             iter != pNodes->end();
             iter++)
        {
            TestNode* pNode = *iter;
            nodes.push_back(pNode);
            struct Functor
            {
                TestNode* pNode;
                Functor(TestNode* val):
                    pNode(val)
                {}
                bool operator()(void)
                {
                    bool testSuccessed = true;
                    TFTRACE("test node \"", pNode->_testCaseName, "\"");
                    if (pNode->_prerequisites.size() == 0)
                    {
                        bool testResult = true;
                        if (pNode->_testCase != NULL)
                        {
                            TestCase* testCase = pNode->_testCase();
                            TFTRACE("test case \"",
                                    testCase->testCaseName(),
                                    "\"");
                            testCase->runTest();
                            testResult = testCase->result;
                            if (false ==  testResult)
                            {
```

```
                    TEST_SUMMARY("test case \"",
                                 testCase->testCaseName(),
                                 "\" failed");
                }
                delete testCase;
                testSuccessed = testResult;
            }
        }
        else
        {
            TEST_SUMMARY("test case \"",
                         pNode->_testCaseName,
                         "\" failed for prerequisites failed");
            testSuccessed = false;
        }
        return testSuccessed;
    }
};
boost::function<bool(void)> func =
    boost::function<bool(void)>(Functor(pNode));
ret.push_back(func);
int index = ret.size() - 1;
do
{
    TestNode* pNode = nodes[index];
    for (std::vector<TestNode*>::iterator iter =
            pNode->_prerequisites.begin();
         iter != pNode->_prerequisites.end();
         iter++)
    {
        TestNode* pChild = *iter;
        nodes.push_back(pChild);
        struct Functor
        {
            TestNode* pChild;
            TestNode* pNode;
            Functor(TestNode* pC, TestNode* pN):
                pChild(pC),
                pNode(pN)
            {
            }
            bool operator() (void)
            {
                TFTRACE("test node \"",
                        pChild->_testCaseName,
                        "\"");
                if (pChild->_prerequisites.size() == 0)
                {
                    bool testResult = true;
                    if (pChild->_testCase != NULL)
                    {
```

```
                                    TestCase* testCase = pChild->_testCase();
                                    TFTRACE("test case \"",
                                            testCase->testCaseName(),
                                            "\"");
                                    testCase->runTest();
                                    testResult = testCase->result;
                                    delete testCase;
                                }

                                if (true == testResult)
                                {
                                    pNode->_prerequisites.pop_back();
                                }
                                else
                                {
                                    TEST_SUMMARY("test case \"",
                                                 pChild->_testCaseName,
                                                 "\" failed");
                                }
                                return testResult;
                            }
                            else
                            {
                                TEST_SUMMARY("test case \"",
                                             pChild->_testCaseName,
                                             "\" failed for prerequisites failed");
                                return false;
                            }
                        }
                    };
                    boost::function<bool(void)> func(Functor(pChild, pNode));
                    ret.push_back(func);
                }
                index++;
            }while(index != ret.size());
        }
    }
    std::reverse(ret.begin(), ret.end());
    return ret;
}
```

⟨Contents of utc_win32gui_test.cpp⟩+≡

```
BOOST_AUTO_TEST_CASE(utc_TestFramework_TestTree_testCaseList)
{
    using OpenGUIL::TestFramework::TestTree;
    TestTree::instance()->registerTestCase(NULL, "rootTestCase", "");
    TestTree::instance()->registerTestCase(NULL, "firstTestCase", "rootTestCase");
    BOOST_CHECK_EQUAL(TestTree::instance()->_testNodes.size(), 2);
    BOOST_CHECK_EQUAL(TestTree::instance()->_rootNodes.size(), 1);
    BOOST_CHECK_EQUAL(TestTree::instance()->testCaseList().size(), 2);
    BOOST_CHECK_EQUAL(TestTree::instance()->testCaseList("rootTestCase").size(), 2);
    BOOST_CHECK_EQUAL(TestTree::instance()->testCaseList("firstTestCase").size(), 1);
    BOOST_CHECK_EQUAL(TestTree::instance()->testCaseList("f").size(), 0);
    TestTree::instance()->clear();
    BOOST_CHECK_EQUAL(TestTree::instance()->_testNodes.size(), 0);
    BOOST_CHECK_EQUAL(TestTree::instance()->_rootNodes.size(), 0);
}
```

8.7 Macros

8.7.1 WIN32GUI_TEST

⟨Head section of win32gui_unit_test.hpp⟩+≡

```
#define WIN32GUI_TEST(name, mustPassBefore)  \
struct name: OpenGUIL::TestFramework::WIN32GUITestCase \
{ \
    static const char* test_case_name; \
    static TestCase* newInstance(); \
    virtual void test(); \
    const char* testCaseName()const {return test_case_name;} \
}; \
const char* name::test_case_name = #name; \
namespace name##__LINE__{ \
struct Helper \
{ \
    Helper() \
    { \
     OpenGUIL::TestFramework::TestTree::instance()->registerTestCase(&name::newInstance, name::test_case_name,
    } \
}; \
Helper h; \
} \
OpenGUIL::TestFramework::TestCase* name::newInstance() \
{ \
    return new name(); \
} \
void name::test()
```

8.8 WIN32CONFIRM

⟨Head section of win32gui_unit_test.hpp⟩+≡

```
#ifdef WIN32
inline bool userConfirm (LPCSTR msg, HWND hWnd)
{
    int ret = ::MessageBox(hWnd, msg, "Confirm", MB_YESNO | MB_ICONQUESTION);
    return ret == IDYES;
}
#endif
```

9 smoke test

⟨CMakeLists.txt⟩+≡

```
add_subdirectory(smoke_test)
```

⟨tangle⟩+≡

```
mkdir smoke_test
notangle -RsmokeTestCMakeList win32gui.nw>smoke_test/CMakeLists.txt
```

⟨smokeTestCMakeList⟩≡

```
include_directories("${PROJECT_SOURCE_DIR}/src")
add_executable(win32guismoketest WIN32 win32guismkt.cpp ../src/win32gui_unit_test.cpp ⟨source files of win32guismoketest
```

⟨tangle⟩+≡

```
notangle -Rwin32guismkt.cpp -t4 -L'#line %L "%F"%N' win32gui.nw>smoke_test/win32guismkt.cpp
```

⟨win32guismkt.cpp⟩≡

```
#include "win32gui_unit_test.hpp"
WIN32GUITESTMAIN
⟨Contents of win32guismkt.cpp⟩
```

⟨Contents of win32guismkt.cpp⟩≡

```
  WIN32GUI_TEST(firstTest, "")
  {
      using std::cout;
      using std::endl;
      cout<<"Hello gui test"<<endl;
      TEST_PASS(true);
  }
  WIN32GUI_TEST(zeroTest, firstTest)
  {
      using std::cout;
      using std::endl;
      cout<<"Hello gui test zero"<<endl;
      static const char kDockWndClassName[] = "DockWndClass";
      HINSTANCE hModule = hInstance;
      TEST_PASS(hModule != NULL);
      TEST_PASS(hWnd != NULL);
      WNDCLASSEX wndClassEx;
      ZeroMemory(&wndClassEx, sizeof(wndClassEx));
      if (FALSE == GetClassInfoEx(hModule, kDockWndClassName, &wndClassEx))
      {
          wndClassEx.cbSize    = sizeof(wndClassEx);
          wndClassEx.style         = CS_HREDRAW | CS_VREDRAW;
          wndClassEx.lpfnWndProc   = DefWindowProc;
          wndClassEx.cbClsExtra  = 0;
          wndClassEx.cbWndExtra  = 0;
          wndClassEx.hInstance   = hModule;
          wndClassEx.hIcon         = NULL;
          wndClassEx.hCursor         = LoadCursor(NULL, IDC_ARROW);
          wndClassEx.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
          wndClassEx.lpszMenuName  = NULL;
          wndClassEx.lpszClassName = kDockWndClassName;
          wndClassEx.hIconSm         = NULL;

          TEST_PASS(0 != RegisterClassEx(&wndClassEx));
      }
      HWND avatarWnd = CreateWindow (kDockWndClassName,//WindowClass
                                      NULL,//WindowTitle
                                      WS_CHILD | WS_VISIBLE,//Style
                                      0,//x
                                      0,//y
                                      480,//width
                                      320,//height
                                      hWnd,//parent
                                      NULL,//menu
                                      hModule,//instance of module
                                      NULL);
      ::ShowWindow(avatarWnd, SW_SHOW);

      TEST_PASS(avatarWnd != NULL);
      TEST_PASS(userConfirm("Click \"YES\" please.", hWnd));
```

```
}

WIN32GUI_TEST(testUserMessage, firstTest)
{
    TEST_MESSAGE("[SMOKE]a smoke message");
    TEST_MESSAGE("a message contains \"[SMOKE]\"");
    TEST_MESSAGE("a user message]");
    TEST_PASS(true);
}
```

10 unit test

⟨CMakeLists.txt⟩+≡

```
add_subdirectory(unit_test)
```

⟨tangle⟩+≡

```
if [ ! -d unit_test ]
then
    mkdir unit_test
fi
chunk=unitTestCMakeLists.txt
target=CMakeLists.txt
dir=unit_test
file=win32gui.nw
notangle -R"$chunk" "$file">"$dir"/"$target"
```

⟨unitTestCMakeLists.txt⟩≡

```
add_executable(utc_win32gui_test utc_win32gui_test.cpp ⟨source files of utc_win32gui_test⟩)
```

10.1 utc_win32gui_test.cpp

⟨tangle⟩+≡

```
if [ ! -d unit_test ]
then
    mkdir unit_test
fi
chunk="utc\\_win32gui\\_test.cpp"
target=utc_win32gui_test.cpp
dir=unit_test
file=win32gui.nw
notangle -R"$chunk" -t4 -L'#line %L "%F"%N' "$file">"$dir"/"$target"
```

⟨utc_win32gui_test.cpp⟩≡

```
#define BOOST_TEST_MAIN
#include <boost/test/unit_test.hpp>
#include "../src/win32gui_unit_test.hpp"
⟨Contents of utc_win32gui_test.cpp⟩
```

11 action

⟨action⟩≡

```
  ⟨tangle_in_linux⟩
  ⟨weave⟩
```

11.1 tangle

⟨tangle_in_linux⟩≡

```
  fileName=win32gui.nw
  file=$fileName
  ltx_file=win32gui.ltx
  ⟨tangle⟩
  ⟨tangle_windows_part⟩
```

⟨tangle_windows_part⟩≡

```
  notangle -R"action\\_in\\_win" -t4 $fileName> action.bat
```

⟨action_in_win⟩≡

```
  ⟨build⟩
  ⟨test⟩
```

11.2 build

⟨build⟩≡

```
  del CMakeCache.txt
  cmake -DCMAKE_MAKE_PROGRAM="devenv" -G"Visual Studio 9 2008" . && devenv win32gui.sln /
  Rebuild Release
```

11.3 test

⟨test⟩≡

```
  IF NOT ERRORLEVEL 0 exit /b -1
  unit_test\Release\utc_win32gui_test.exe
  smoke_test\Release\win32guismoketest.exe --trace --should-confirm-quit --filter-user-message="^
  \[SMOKE\]"
  ⟨clean⟩
```

11.4 packed

⟨clean⟩≡

```
  IF NOT ERRORLEVEL 0 exit /b -1
  rd /S /Q CMakeFiles
  rd /S /Q Release
  rd /S /Q Debug
  rd /S /Q testframework.dir
  rd /S /Q ZERO_CHECK.dir
  del cmake_install.cmake
  del CMakeCahe.txt
```

11.5 weave

⟨weave⟩≡

```
noweave -option shift -option smallcode $file| \
sed 's/\\usepackage{noweb}/\\usepackage[top=1.2in,bottom=1.2in,left=1.2in,right=1in]{geometry}
&/g'| \
sed 's/\\usepackage{noweb}/\\usepackage{fontspec, xunicode, xltxtra}&/g'| \
sed 's/\\usepackage{noweb}/\\usepackage{listings}&/g'| \
sed 's/\\usepackage{noweb}/\\usepackage[120, ampersand]{easylist}&/g'| \
sed 's/\\usepackage{noweb}/\\usepackage{paralist}&/g'| \
sed 's/\\usepackage{noweb}/\\usepackage{color}&/g'| \
sed 's/\\usepackage{noweb}/\\usepackage{hyperref}&/g'| \
sed 's/\\usepackage{noweb}/\\usepackage{underscore}&/g'| \
sed 's/\\usepackage{noweb}/&\\XeTeXlinebreaklocale "zh-cn"/g'| \
sed 's/\\usepackage{noweb}/&\\pagecolor{grayyellow}/g'| \
sed 's/\\usepackage{noweb}/&\\definecolor{grayyellow}{RGB}{255, 255, 200}/g'| \
sed 's/\\usepackage{noweb}/&\\XeTeXlinebreakskip = 0pt plus 1pt minus 0.1pt/g'| \
sed 's/\\usepackage{noweb}/&\\setmainfont[BoldFont={Adobe Heiti Std}]{Adobe Song Std}/g'| \
sed 's/\\begin{document}/&\\tableofcontents/g'| \
sed 's/\\documentclass[11pt]/&[11pt]/g'|
sed 's/  /  /g'> $ltx_file
xelatex $ltx_file
xelatex $ltx_file
echo $ltx_file|sed 's/ltx$/log/g'|xargs rm -rf
echo $ltx_file|sed 's/ltx$/aux/g'|xargs rm -rf
echo $ltx_file|sed 's/ltx$/toc/g'|xargs rm -rf
echo $ltx_file|sed 's/ltx$/out/g'|xargs rm -rf
rm $ltx_file
```