flores V 4.0
date Sep-2011

1 Introduction

The ***flores*** library implements classes that are helpful for virtually every C++ project.

It's origin is in the 1990-ies; that explains that there is a String class - since at that time there was no STL string available (at least not on Sun Solaris).

And it explains that ***flores*** has it's own test framework (nowadays one would use google gtest ...)

1.1 package Overview

flores contains the following packages (e.g. namespaces) :

flores::diagnostics
flores:: lang
flores:: net
flores:: io
flores:: util

1.2 Design

Version 4 (2011) was completely revamped to make the library more Java like

- Class names and methods are the same as in Java (wherever applicable)

- We like the class import clause in java, so we crated an include directory structure to mimick import syntax :
  #include <flores/net/Socket.h> looks close to
  import java.net.Socket :-)

- We like the ability of Java IDE's to structure the file list as a tree - sorted by package. To achieve this in a C++ IDE like MSVC, source filenames include their package (e.g. namespace)

  the file-list in the IDE then looks as follows

```
...
flores.net.ServerSocket
flores.net.Socket
...
flores.util.CTime
...
```

1.3 Supported Compilers

- Windows : Visual Studio 2008 and later
- Linux : gcc V3 and later

1.4 Supported Operating systems

- Windows 32 bit
- Linux / Unix

1.5 License

flores is is open source and freely available under Apache License 2.0
(www.apache.org/licenses/LICENSE-2.0)

Contributor(s) :
CTS is open source and freely available under Eclipse Public License 1.0.

1.6 More Information

Class Reference Guide (created with DoxyGen)
Tests : can be regarded as learning tests :-)
Examples

1.7 Disclaimer

This documentation is work in progress !

2 Setup

Unzip the distribution to any directory.

Example : let's assume you 'installed' ***flores*** in 'X:\development\libraries\flores'

This directory will from now on be referred to as the ***flores-root***

2.1 Sources

add all cts* sources	from ***flores-root***\contrib\cts
add all flores sources	from ***flores-root***\src
optional : if you want to test flores	
add tests	from ***flores-root***\test\src

2.2 Project Settings

flores is **not** written as managed code (otherwise it won't run on Linux), so make sure you create an WIN32 ('umanaged code') project.

- WIN32
  Define this when compiling on WIN32
  ⌦ Is usually pre-defined by the Windows IDE

- __FLORES_EXPORT__
  define this when you want to create a DLL from the flores sources.

2.2.1 additional include directories

add the following directories to the include path :
flores-root/include
flores-root/include/cts

optional : if you want to test flores, add :
flores-root/test/include

2.2.2 character set (windows only)

General
  Project Defaults
    Character set
      use **MULTI BYTE** character set

2.2.3 required libraries

- On UNIX, add libraries **rt** and **pthread**
- On Windows, using Visual C++, make sure you add the library **WS2_32.lib** (winsock version 2).
  ⌦ This library is only required when you use classes from ***flores.net***

2.2.4 optional libraries

- flores.net.ReadWriteTest
  this test uses the JTC (Java Threads for C) library.
  If you want to run it, you need to :

  a) add JTC sources (available from ftp.dreamtime.org/pub/programming/c++/orbacus-jtc/2.0/) to your project
  b) define __FLORES_HAVE_JTC__
  c) add the directory 'above' the JTC installation directory to the include path.
  Example : let's assume you 'installed' JTC in 'X:\development\libraries\JTC', then add 'X:\development\libraries' to the include path.

3 net package

3.1 Overview

Some sections of a client- or server program using the socket API can be difficult to understand for novice socket programmers.
In addition, these sections are often repetitive , e.g. they are copied and pasted from one application to the next.

In addition, support for TELNET clients raises some subtle issues. Yes you are reading right : TELNET !
Reason : Students proudly finish their first Client-Server project, alas, at runtime it won't behave as expected. Who is the culprit - the Client or the Server ? To answer this question, students are advised to use TELNET as a Client : If the behaviour is still not as expected - it's the Server's fault ...

Last not least, (only) on the Windows platform, the socket DLL must be initialized properly.

3.2 Server Example

A simple Echo Server.

⌦ not all lines of code are shown.

⌦ full source code can be found in the examples directory of the source code distribution

```
/**
 **  file              : EchoServer.cpp
 **  description : minimalistic implementation - terminates after
 **                       satisfying the first request.
 **  Conclusion  : not very useful, just for demonstration.
**/
...
int main()
{
        flores::net::Socket*                    pSocket;
        char                                    line [MAXLINE];
        int                                     lineLen;
        unsigned short                          port = ECHO_PORT;
        flores::net::ServerSocket               ss;

        if ( ! ss.bind (port))
return 1;

        pSocket = ss.accept ();

        bzero (line, sizeof line);

        lineLen = pSocket->read (line, sizeof line);

        pSocket->write (line, lineLen);
        delete (pSocket);
        return 0;

}
```

3.3 Client Example

A simple Echo Client

⌦ not all lines of code are shown.

⌦ full source code can be found in the examples directory of the source code distribution

```
int main(int argc, char * argv[])
{
        unsigned short                    port = ECHO_PORT;
        char                              servername [80];
        char                              inBuff  [MAXLINE];
        char                              outBuff  [] = "hello";
         flores::net::Socket              sock;


         if ( ! sock.connect ( servername, port))
return 1;


        if ( sock.write (outBuff, strlen(outBuff)) == -1)
return 2;


        if (sock.read (inBuff, sizeof inBuff)  < 0)
return 3;

        return 0;
}
```

3.4 EOL

End of Line indicators vary netween operating systems and applications ...

• <CR><LF>	: Windows style
• <LF>	: UNIX style
<LF> is used by :	TELNET SMTP, HTTP, ... Java Network Applications
• '\0' may be used by	an Application implemented in C/C++, sending strings wich already contain a terminating Null

The Socket.readLine() method checks for all these EOL indicators.

4 util package

4.1 Properties class

Eases the task of reading and writing INI files.
Without this class one would have to write code like this, which is not very convenient ...:

```
        char key   [  ] = "Nickname";
        char value [80] = { '\0' };

        int valueLen = GetPrivateProfileString (
                "Client"                    // section
               ,key                         // key
               ,""                          // Default,
               ,value                       // ReturnedString,
               ,sizeof(value)               // Size of ReturnedString
               ,".\\ChatClient.ini"   );    // INI File Name

           if (valueLen ==0)
                cout << "INI file  or section  or key  not found!" << endl;
        else cout << "value of key " << key << " is " << value << endl;
    }
```

5 dsl package

Provides a (Generic) LexicalAnlyzer and implementations for C++ and LegalNumbering.

We are aware that nowadays one would use frameworks like antlr (for Java) etc., yet we maintain these classes - maybe out of 'nostalghia' :-)

6 Throwables and Test support

6.1 Design

Design of Throwables is inspired by the concept of preconditions and postconditions (B. Meyer).
flores classes implement preconditions and postconditions.

6.2 Classes

flores::lang::***Throwable*** is the parent for all throwables.
Throwable has two direct subclasses :
Slip : When a precondition is violated, a ***Slip*** is thrown (because you, the user, did something wrong, e.g. you slipped).
Panic : When a Postcondition is violated, a ***Panic*** is thrown (because we, the developers, did something wrong).

Creating an application-specific set of expressive Slips and Panics is good practice (as in Java).

- Available Slips

  lang::NumberFormatSlip
  lang::IllegalArgumentSlip
  lang::ArrayIndexOutOfBoundsSlip
  io::FileExistSlip

- Available Panics

  lang::BufferTooSmallPanic
  lang::OutOfMemoryPanic
  io::FileOpenPanic
  io::FileCreatePanic
  io::FileRenamePanic
  io::FileDeletePanic
  net::CannotCreateSocketPanic
  net::ProtocolPanic

6.3 macros for Testers

- ***assertTrue*** (condition)
- ***assertFalse*** (condition)

- ***assertSlipped*** (method-call)
  Verify that we violated a precodition, for example by calling a method with invalid argument(s)

  Example : assertSlipped(myBank.withdrawal (100000000));

6.4 class for Testers

- Statistician
  call Statistician.begin (<TestName>) at the start of your test and and Statistician.end (<TestName>), and Statistician will log an passed or failed message.

  ⌦ In a future release of flores, Statistician will count the number of passed and failed asserts.

6.5 Example

```
void main ()
{
        Statistician::begin ("BankAccount");
        double initialBalance = 100.0;  double amount  = 50.0;

        BankAccount ba (initialBalance);
         ba.withdrawal (amount);
        assertTrue ( ba.getBalance() ==  initialBalance - amount );
        ba.deposit(amount);
        assertTrue ( ba.getBalance() == initialBalance );

        initialBalance =  50.0;        too_much  = 100.0;

        BankAccount ca (initialBalance);
        assertSlipped ( ca.withdrawal (too_much));
        assertSlipped ( ca.deposit ( - amount ));

       Statistician::end ("BankAccount");
}
```

6.6 Example output

```
INFO    ==== Test for BankAccount begins=============================

FINE    line  30 :ok: assert True ( ba.getBalance() == initialBalance - amount)
FINE    line  33 :ok: assert True ( ba.getBalance() == initialBalance )
FINE    Insufficient funds in account 2
             Balance was 50.
             Withdrawal was 100.
FINE    line  44 ok: assert Slip (ca.withdrawal(amount))
FINE    >amount > 0< I:\flores\examples\diagnostics\bankaccount.cpp # 48
FINE    line  47 :ok: assert Slip (ca.deposit(-amount))
INFO    ==== Test for BankAccount ok  ================================
```

6.7 macros for developers

- ***require*** (condition)
  If the requirement is not met, this macro throws a ***Slip***.
  nB : require is a keyword in Eiffel (the language created by B. Meyer)

  🕮 Example

```
void BankAccount::deposit (double amount)
{
        require  (amount > 0)
        balance += amount;
}
```

- ***ensure*** (condition)
  If the condition is false, this macro throws a ***Panic***.
  nB : ensure is a keyword in Eiffel (the language created by B. Meyer)

  🕮 Example

```
void String::insert (const String s, int pos)
{
        ....
        ensure (isConsistent())
}
```

- ***[i]raise***
  If you want to throw a sub-class of Slip or Panic, you could use **throw**.

  However, using the macro***s***

raise (ThrowableClassName, message) or
iraise (ThrowableInstance)
has advantages :

[i]raise stores additional information (Filename, line-number) where the Slip or Panic was thrown.