Module : ConfigurationMgr (libConfigurationMgr.so)

Author : Kabir Sohel

Version : 1.0

Date : 02 Oct 2012

Dependency : pthread

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Configuration Manager has two parts :

* + - Configuration
      * Loads configuration data from file
      * Exposes APIs to access / modify configuration data.
      * Listens to File Monitor for file change event.
      * Triggers the listeners when configuration changes. (optional)
      * Update Configuration Data when file changes.
    - File Monitor
      * Monitors Files constantly.
      * Exposes APIs to control monitoring interval
      * Triggers the listeners when file change occurs.

How To Use:

ConfigurationMgr::Instance()->startConfigurationMgr();

try {

ConfigurationMgr::Instance()>addPropertyFile(3,"file1.txt","file2.txt","file3");

}

catch (ConfigurationException \*exception){

cout << exception->getMessage() << endl;

}

addPropertyFile() also has a overloaded addPropertyFile(const string &file) method.

To use an api simply call ConfigurationMgr::Instance()>apiName()

Configuration Manager inherits from Configuration and use FileMonitor class. ConfigurationMgr has the following methods including the inherited protected and public methods from Configuration class.

static ConfigurationMgr\* Instance();

void startConfigurationMgr();

void addPropertyFile(int numberOfFiles, ...);

void addPropertyFile(const string &fileName);

void removePropertyFile(const string &fileName);

void parseRawInput(std::vector<std::string> &data, const string &fileName);

//overriden methods from Configuration

void loadPropertyConfig(const string &fileName);

void loadXMLConfig(const string &fileName);

Configuration : Data Structures :

vector < string > fileNames;

// Data holds the key value pair

// each file maps with its own set of data.

struct Data

{

string key;

string value;

Data(std::string \_key, std::string \_value)

{

key = \_key;

value = \_value;

}

bool operator<(const Data &ob)const{return this->key < ob.key;}

};

map < std::string , set < Data > > files;

list<ConfigurationEventListener \*> listeners;

typedef enum

{

CONFIGURATION\_ADDED,

CONFIGURATION\_DELETED,

CONFIGURATION\_MODIFIED,

}EventType;

//event will be triggered when application listens to the dynamic update of changes.

class ConfigurationEvent

{

private:

EventType eventType;

string key; //optional

string value; //optional

string fileName;

};

Configuration : Method Lists :

private:

void update(ConfigurationEvent event);

void fileChanged(const string &fileName);

public:

Configuration();

virtual ~Configuration();

protected:

void addSource(const string &fileName);

//void addSource(int numberOfParam, ...);

void removeSource(const string &fileName);

public:

//methods to be implemented by child class

virtual void loadPropertyConfig(const string &fileName) = 0;

virtual void loadXMLConfig(const string &fileName) = 0;

//Method implemented for File Monitor

void fileChanged(FileMonitorEvent event);

// public APIs

void addProperty(const string &key, const string &value);

void addProperty(const string &fileName, const string &key, const string &value);

void deleteProperty(const string &key);

void deleteProperty(const string &key, const string &value);

void deletePropertyFromFile(const string &fileName, const string &key);

void deletePropertyFromFile(const string &fileName, const string &key, const string &value);

void deleteAll();

void deleteAllFromFile();

void deleteAllFromFile(const string &fileName);

bool isEmpty();

void addListener(ConfigurationEventListener\* listener);

void removeListener(ConfigurationEventListener\* listener);

void removeAllListener();

bool hasData(const string &fileName, const Data &data);

bool hasKey(const string &key);

bool hasKeyInFile(const string &fileName, const string &key);

bool hasValue(const string &key);

bool hasValueInFile(const string &fileName, const string &key);

bool hasValue(const string &key, const string &value);

int countKey();

int countValue();

int countValue(const string &key);

//get all the keys Configuration has.

set<string> getKeyAsSet();

list<string> getKeyAsList();

vector<string> getKeyAsVector();

//get values associated to one key or a set of key

string get(const string &key);

string getValueFromFile(const string &fileName, const string &key);

string getValueAsString(const string &key); //returns the first value if multiple

set <string> getValueAsSetOfString(const string &key);

list <string> getValueAsListOfString(const string &key);

vector<string> getValueAsVectorOfString(const string &key);

bool getValueAsInt(const string &key,int &intValue);

vector<int> getValueAsVectorOfInt(const string &key);

string replaceVariable(const string &value);

Configuration : Key Notes :

* + Configuration key value pair is separated by a single “=”.
  + Initial spaces and trailing spaces is removed while processing.
  + One key may have multiple values in different file.
  + If one key has multiple values, the value contained in the file which was added at first will be returned.
  + User can also get the all values associated with a key. Look at the API list.
  + addPropertyConfig () is a variable argument method , first param is the number of file to be added followed by the files separated by comma.
  + Get always returns with recursive variable replacement when “$” is found. for example, if the configs are like the following
    - Port = 8080
    - IP = 127.0.0.1
    - host = <http://$IP:$Port>

the get(“host”) will result <http://127.0.0.1:8080> . get() will try to replace variable as much as it can. To omit variable replacing add a escape character before “$” symbole. Ie. host = <http://\$IP.com:$Port>

* + If the application wants to get the dynamic effect the configuration data change, then it should have one class that implements ConfigurationEventListener interface to get the

void configurationChanged(ConfigurationEvent event) method triggered. event will have the configuration key value that’s changed.

* + Application may have a number of files. All of the config data must be text based equal separated.
  + If any property of the file is deleted then it will notify the listeners (if has) . but it will keep the configuration data as a default configuration. If however in the future that configuration is added to the file , default configuration will be overwritten.
  + XML Configuration and DB Configuration will be implemented later.

File Monitor : Data Structures :

typedef enum{

FILE\_CHANGED,

FILE\_ADDED,

FILE\_DELETED,

FIEL\_PERMISSION\_RESTRICTED,

}FileChangeEventType;

class FileMonitorEvent

{

public:

FileChangeEventType eventType;

string fileName;

FileMonitorEventListener\* eventListener;

FileMonitorEvent(){}

FileMonitorEvent(std::string fileName, FileChangeEventType eventType,

FileMonitorEventListener \*listener){

this->eventType = eventType;

this->fileName = fileName;

this->eventListener = listener;

}

}

list <FileMonitorEventListener\*> listeners;

static FileMonitor \*instance;

struct File

{

File(std::string \_file, long long \_intervalInMSecond, bool \_monitoring)

{

fileName = \_file;

monitoring = \_monitoring;

intervalInMSecond = \_intervalInMSecond;

}

std::string fileName;

bool monitoring;

long long intervalInMSecond;

long long lastTimeStamp;

struct stat st;

Timer timer;

pthread\_t pthread;

int threadId;

FileMonitorEvent event;

};

vector <File> files;

File Monitor : API Lists :

static FileMonitor\* getInstance();

int getFileLocation(std::string fileName);

static void \*fileMonitorLoop(void \*param);

bool isFileChanged(int fileLocation);

bool addFile(std::string fileName, long long mSec = 100, bool dontStartNow = false);

string addFile(int numberOfParam, ...);

bool isFileAdded(std::string fileName);

void removeFile(std::string fileName);

void addListener(FileMonitorEventListener\* listener);

void removeListener(FileMonitorEventListener\* listener);

bool isListenerAdded(FileMonitorEventListener\* listener);

void removeAllListeners();

void stopMonitoring(std::string fileName);

bool startMonitoring(std::string fileName);

bool isBeingMonitored(std::string fileName);

bool canMonitor(std::string fileName);

void resetFileMonitorInterval(std::string fileName, long long mSec);

long long getFileMonitorInterval(std::string fileName);

File Monitor : Key Notes :

* + To use Configuration Manager user doesn’t need to bother about File Monitor. Configuration Manager does everything. Still if the user wants to access the api, he/she is welcome to use, Its singleton.
  + addFile has overloaded method which can take variable argument.
  + Default file change check interval is 100ms . user can change it at the time of calling addFile or resetFileMonitorInterval.
  + Each file has separate thread to notify listeners only when it changes.

Class Diagram

