Configuration Files

Configuration file support in the Util library.
Overview

- AbstractConfiguration
- INI Files
- Property Files
- XML Files
- Other configuration data sources
Poco::Util::AbstractConfiguration provides a common interface for accessing configuration information from different sources.

Configuration settings are basically key/value pairs, where both key and value are strings.

Keys have a hierarchical structure, consisting of names separated by periods.

Values can be converted to integers, doubles and booleans.

An optional default value can be specified in the getter functions.
AbstractConfiguration Members

- bool hasProperty(const std::string& key)
- std::string getString(const std::string& key [, const std::string& default])
- int getInt(const std::string& key [, int default])
- getDouble(), getBool()
- setString(), setInt(), setDouble(), setBool()
- keys()
ConfigurationViews

> **ConfigurationView** allows you to create a "view" into a sub hierarchy of another configuration.

> Say, you have:
config.value1, config.value2, config.sub.value

> Create a view on prefix config, the in the view, you have value1, value2, sub.value
INI Files

- `Poco::Util::IniFileConfiguration` supports plain old INI files, as used mostly on Windows.
- Key names are not case sensitive.
- Leading and trailing whitespace is removed from both keys and values.
- read-only
using Poco::AutoPtr;
using Poco::Util::IniFileConfiguration;

AutoPtr<IniFileConfiguration> pConf(new IniFileConfiguration("test.ini"));

std::string path = pConf->getString("MyApplication.somePath");
int value = pConf->getInt("MyApplication.someValue");
value = pConf->getInt("myapplication.SomeValue");
value = pConf->getInt("myapplication.SomeOtherValue", 456);
Property Files

- Property Files are known mainly from Java.
- Key names are case sensitive.
- The backslash is used for escaping, so be careful when specifying Windows path names.
- writable
key1 = value1
def key2: 123
def key3.longValue = this is a very \
  long value
def path = c:\\test.dat

using Poco::AutoPtr;
using Poco::Util::PropertyFileConfiguration;

AutoPtr<PropertyFileConfiguration> pConf;
pConf = new PropertyFileConfiguration("test.properties");

std::string key1 = pConf->getString("key1");
def value = pConf->getInt("key2");
def longVal = pConf->getString("key3.longValue");
XML Configuration Files

- XML configuration files are parsed with the DOM parser and thus fully loaded into memory.

- Both text in elements, as well as attribute values can be accessed, using a XPath-like syntax.

- writable (fully writable since 1.3.4)
using Poco::AutoPtr;
using Poco::Util::XMLConfiguration;

AutoPtr<XMLConfiguration> pConf(new XMLConfiguration("test.xml"));

std::string prop1 = pConf->getString("prop1");
int prop2 = pConf->getInt("prop2");
std::string prop3 = pConf->getString("prop3"); // ""
std::string prop4 = pConf->getString("prop3.prop4"); // ""
prop4 = pConf->getString("prop3.prop4[@attr]"); // "value3"
prop4 = pConf->getString("prop3.prop4[1][@attr]"); // "value4"
Other Configurations

- **FilesystemConfiguration**: a separate file for each configuration property, stored in a directory hierarchy.

- **LayeredConfiguration**: allows layering of multiple configurations

- **MapConfiguration**: stored in a `std::map<std::string, std::string>`

- **SystemConfiguration**: system.osName, system.osVersion, system.currentDir, etc.

- **WinRegistryConfiguration** (Windows only)
LayeredConfiguration

- A LayeredConfiguration consists of a number of AbstractConfiguration instances.
- When reading a configuration property, all added configurations are searched, in order of their priority.
- Configurations with lower priority values have precedence.
- When setting a property, the property is always written to the first writeable configuration (see addWriteable()).
- If no writeable configuration has been added to the LayeredConfiguration, and an attempt is made to set a property, a RuntimeException is thrown.
LayeredConfiguration (cont'd)

- Every configuration added to the `LayeredConfiguration` has a priority value (`int`).

- The priority determines the position where the configuration is inserted, with lower priority values coming before higher priority values.

- If no priority is specified, a priority of 0 is assumed.