Dr.Dobb's
THE WORLD OF SOFTWARE DEVELOPMENT

Tech Tips

Source Code Accompanies This Article. Download It Now.

- tt0404.txt

Gigi Sayfan and Matthew Wilson show how to share the clipboard and use Managed C++ strings.

April 01, 2004
URL:http://www.drdobbs.com/cpp/tech-tips/184405624

George is a software engineer in the System Design and Verification group at Cadence Design Systems. He can be reached at georgefrazier@yahoo.com.

Sharing the Clipboard

by Gigi Sayfan

the_gigi@hotmail.com

The little program I present here lets multiple computers on a LAN share a clipboard. This is useful in situations where a single person is operating multiple computers at the same time. Think about developing and testing client-server applications, administering several machines through a KVM switch, or simply using one machine for Internet browsing while working on the other. You can even use it for ad hoc instant messaging.

The trick is to use a shared file as a repository for clipboard content. The ShareClipboard programs that run on each computer monitor both this file and the real clipboard. If the file content has been modified, it means that the clipboard content on some other machine has changed and all the monitoring machines will synchronize. If the clipboard content has changed, the program will place the new content on the shared file for other machines to synchronize.

I provide two functionally equivalent versions: Python (Listing One) and C/C++ (Listing Two) (compiled using Visual C++.NET 2003). You will need the win32extensions modules for Python to run the Python version.

You can download a free and fully functional Python implementation from ActiveState at http://www.activestate.com/Products/ActivePython/.

Accessing C-String Representations of Strings in Managed C++

by Matthew Wilson

matthew@synesis.com.au

In Managed C++, you can create instances of a .NET *System::String* class from C++ C-style strings (null-terminated *char/wchar_t const* *). However, getting a null-terminated C-style string from a *String* is much less simple. Since *String*s may not use the same encoding that you want, or store the characters contiguously, or be null terminated, it is not as simple as returning a pointer.

If you have the spare time, you can create a *char/wchar_t* buffer of the appropriate length, and then use the *Chars* property (the *String::get_Chars()* method in Microsoft C++) to retrieve each character, thereby manually building you a C-string. Tedious.

The .NET libraries have thought about this for you, and provided you with the *System::Runtime::InteropServices::Marshal* methods *StringToHGlobalAnsi()*, *StringToHGlobalUni()*, and *FreeHGlobal()*, which are used as in Listing Three(a). The pointer returned from *StringToHGlobalAnsi()* is not part of the managed heap and must be free, by the call to *FreeHGlobal()*. This is classic resource-scoping, where the native pointer may be leaked if an exception is thrown before it is freed, and is crying out for a touch of Resource Acquisition Is Initialization.

The STLSoft subproject .netSTL (http://dotnetstl.org/) contains just such a scoping class, in the form of the template c_string_accessor, defined in Listing Four. This simplifies the code (both in terms of lines of code, and in removing the need to "use" the *System::Runtime::InteropServices::Marshal* type), as well as handling exceptional circumstances. Listing Three(a) changes to Listing Three(b).

DDJ

Listing One

```
#!/usr/local/bin/python
import time
from win32clipboard import *
```

```
id = 0
clipboard_file = r'\\home\Clipboard\clipboard.txt'
prev_data = ''

while (True):
   time.sleep(1)
   if OpenClipboard() != None:
      print 'OpenClipboard() failed'
      continue
   try:
      data = prev_data
      if    IsClipboardFormatAvailable(CF_TEXT):
         data = GetClipboardData()
      if data != prev_data:
         open(clipboard_file, 'w').write(data)
         print 'writing %s to file' % data
         prev_data = data
      else:
         data = open(clipboard_file, 'r').read()
         if data != prev_data:
            EmptyClipboard()
            SetClipboardData(CF_TEXT, data)
            print 'putting %s in clipboard' % data
            prev_data = data

      CloseClipboard()
   except:
      pass
```

Back to Article

Listing Two

```
#include <fstream>
#include <iostream>
#include <string>

int APIENTRY _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                       LPTSTR lpCmdLine, int nCmdShow)
{
   const char * clipboard_file = "\\\\home\\Clipboard\\clipboard.txt";
   std::string prev_data;
   std::string data;
   while (true)
   {
      ::Sleep(1000);
      if (::OpenClipboard(0) == FALSE)
         continue;
      data = prev_data;
      if (::IsClipboardFormatAvailable(CF_TEXT) != FALSE)
      {
         HANDLE h = ::GetClipboardData(CF_TEXT);
         if (h)
         {
            const char * buf = (const char *)::GlobalLock(h);
            data = buf;
            if (!data.empty())
               ::GlobalUnlock(h);
         }
      }
      if (data != prev_data)
      {
         std::ofstream f(clipboard_file);
         if (!f.is_open())
         {
            ::CloseClipboard();
            continue;
         }
         f << data;
         f.close();
         prev_data = data;
      }
      else
      {
         std::ifstream f(clipboard_file);
         if (!f.is_open() || f.eof())
         {
            ::CloseClipboard();
            continue;
         }
         data = "";
         char c;
         f.get(c);
         while(!f.eof())
         {
            data += c;
            f.get(c);
         }
         f.close();
         if (data != prev_data)
         {
```

```
            BOOL rc = ::EmptyClipboard();
            if (rc == FALSE)
            {
               ::CloseClipboard();
               continue;
            }
            HGLOBAL hMem = ::GlobalAlloc(GMEM_MOVEABLE, data.length()+1);
            void * buf = ::GlobalLock(hMem);
            ::memcpy(buf, (const void *)data.c_str(), data.length()+1);
            ::GlobalUnlock(hMem);
            HANDLE h = ::SetClipboardData(CF_TEXT, hMem);
            if (h == NULL)
            {
               ::CloseClipboard();
               continue;
            }
            prev_data = data;
         }
      }
      ::CloseClipboard();
   }
}
```

Back to Article

Listing Three

(a)

```
using System::String;
using System::Runtime::InteropServices::Marshal;
String  *s = ... // get a string from somewhere
char  *ansi = (char*)(Marshal::StringToHGlobalAnsi(s).ToPointer());
puts(ansi);
Marshal::FreeHGlobal(ansi);
```

(b)

```
using dotnetstl::c_string_accessor;
String  *s = ... // get a string from somewhere
puts(c_string_accessor<char>(s));
```

Back to Article

Listing Four

```
template <typename C>
class c_string_accessor
{
public:
  typedef C                     char_type;
  typedef C                     *pointer;
  typedef C const               *const_pointer;
  typedef c_string_accessor<C>  class_type;

// Construction
public:
  ///
  explicit c_string_accessor(System::String *s)
    : m_s(_get_cstring(s))
  {}

  ~c_string_accessor()
  {
    System::Runtime::InteropServices::Marshal::FreeHGlobal(m_s);
  }

// Implementation
public:
  operator const_pointer() const
  {
    return m_s;
  }

// Implementation
private:
  pointer _get_cstring(System::String *s);

// Members
private:
  pointer m_s;

// Not to be implemented
private:
  c_string_accessor(class_type const &);
  c_string_accessor &operator =(class_type const &);
};

// Specialization for char
template <>
```

```
inline c_string_accessor<char>::pointer
    c_string_accessor<char>::_get_cstring(System::String *s)
{
  return reinterpret_cast<char*>(System::Runtime::
      InteropServices::Marshal::StringToHGlobalAnsi(s).ToPointer());
}

// Specialization for wchar_t
template <>
inline c_string_accessor<wchar_t>::pointer
    c_string_accessor<wchar_t>::_get_cstring(System::String *s)
{
  return reinterpret_cast<wchar_t*>(System::Runtime::
      InteropServices::Marshal::StringToHGlobalUni(s).ToPointer());
}
```

Back to Article