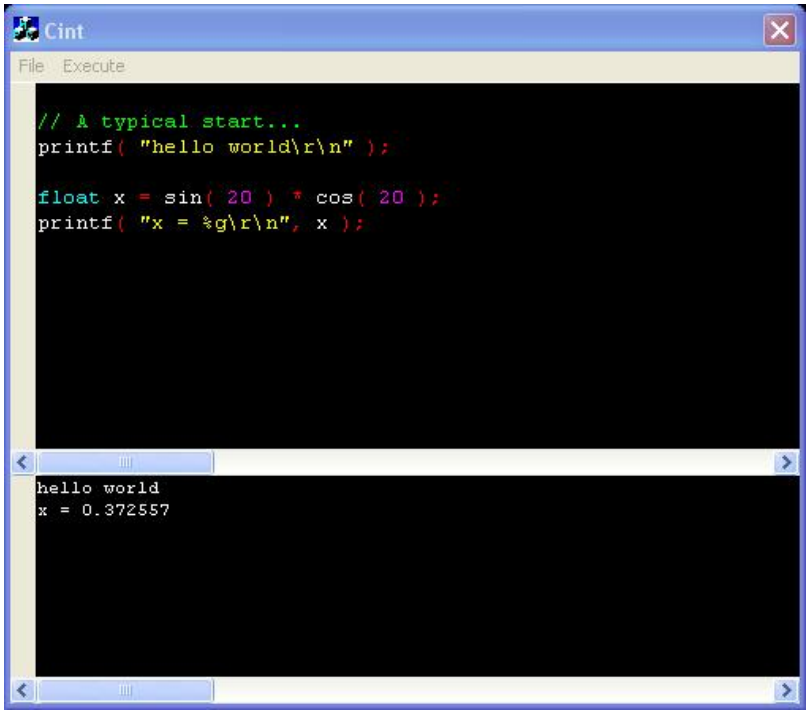
Building a simple C++ script compiler from Scintilla and CINT

By **Robert Umbehant** | 8 Jul 2006

Licence	CPOL
First Posted	**8 Jul 2006**
Views	**54,583**
Bookmarked	**63 times**
Tags	VC6, VC7, VC7.1, VC8.0, Win2K, WinXP, ...

How to build a simple C++ script compiler from Scintilla and CINT.

Download demo - 694 KB
Download source - 3.34 MB



Introduction

I ran across two Open Source projects recently. CINT, a C/C++ script engine, and Scintilla, a multi-language editor with syntax highlighting among other features.

So the possibilities here are obvious. A simple application that would allow the rapid entry, execution, and debugging of C/C++ scripts. What a handy tool for a developer such as myself. I would imagine this would also be of interest to anyone learning the C/C++ language.

Application Goals

There is tremendous opportunity here for features. But I decided to keep the feature set as small as possible so as to focus on the integration of the CINT engine and the Scintilla editor.

The features provided will enable you to enter your C/C++ code into the editor, execute it, then the program will rub your nose in all the numerous syntax errors you have inflicted on the digital world.

This application also statically links both projects, and I will cover the details needed to make this happen. Often with small applications like this, you know, without full blown installers, it's nicer to static link to avoid frustrating users with missing DLLs. My personal preference is to static link if possible unless I have an installer that is capable of verifying the integrity of the installation. Disk space is cheaper than time these days.

Our project

The project is an MFC dialog application created with VC6. I avoided using MFC specific classes for integrating CINT or Scintilla, so you should have no problem porting this code to a non-MFC project.

Linking to Scintilla

From the Scintilla website - *Scintilla is an editor control providing support for syntax styling, error indicators, code completion, and call tips. The selection margin can contain markers like those used in debuggers to indicate breakpoints and the current line. Styling choices are more open than with many editors, allowing the use of proportional fonts, bold and italics, multiple foreground and background colors, and multiple fonts.*

I downloaded and extracted Scintilla into the sub folder *scintilla* in our project folder. We find the VC project files in the sub folder *./scintilla/vcbuild*. After adding the *SciLexer.dsp* project to our workspace, we find it builds without error. Great!

By default, Scintilla compiles to a DLL. We would like to static link, so we will add a **linker response file** to create a static library. I created two files, one for the release version, and another for the debug version.

Linker Response File (*rsp_scintilla.txt*) - Release version.

Collapse | Copy Code

```
/nologo /subsystem:windows ./Release/*.obj /out:../bin/s_scintilla.lib
```

Linker Response File (*rsp_scintillad.txt*) - Debug version.

Collapse | Copy Code

```
/nologo /subsystem:windows ./Debug/*.obj /out:../bin/sd_scintilla.lib
```

Now we add a **Post-Build Step** to each of the Release and Debug versions, calling the appropriate response file.

Post-Build Step - Release version.

Collapse | Copy Code

```
link -lib @rsp_scintilla.txt
```

Post-Build Step - Debug version.

Collapse | Copy Code

```
link -lib @rsp_scintillad.txt
```

Build Scintilla, and you should find that the files *sd_scintilla.lib* and *s_scintilla.lib* have been created in the *scintilla/bin* folders. These are the libs we will link to.

We need to add Scintilla headers to our project, so I prefer to do this in the *Stdafx.h* file, since the Scintilla library files will probably not change much. So, in *Stdafx.h*, let's add the following includes...

Collapse | Copy Code

```
// Include Scintilla parser
#include "scintilla/include/SciLexer.h"
#include "scintilla/include/Scintilla.h"
```

Last step here, we need to link to the Scintilla lib files. Open the *Stdafx.cpp* file and add the following....

Collapse | Copy Code

```
#ifdef _DEBUG
#   pragma comment( lib, "scintilla/bin/sd_scintilla.lib" )
#else
#   pragma comment( lib, "scintilla/bin/s_scintilla.lib" )
#endif
```

This will link us to the debug or release version as needed. You could also add the lib files to the project settings, I prefer this method.

Since the Scintilla project was a DLL, and we're now linking statically, we need to reproduce the startup and shutdown procedures it would have done in the DLL. You can usually find this by searching the project for the function `DllMain()`. Sure enough, we find `DllMain()` in Scintilla. It registers the editor window class among other things. We will add this code to our startup. In `InitInstance()`, add...

Collapse | Copy Code

```
// Initialize the Scintilla
if ( !Scintilla_RegisterClasses( AfxGetApp()->m_hInstance ) )
{   AfxMessageBox( "Scintilla failed to initiailze" );
    return FALSE;
} // end if
```

Add the Scintilla shutdown code to `ExitInstance()`...

Collapse | Copy Code

```
// Release Scintilla
Scintilla_ReleaseResources();
```

Great! We are now linked to the Scintilla library. Let's move on to CINT.

Linking to CINT

From the CINT website - *CINT is a C/C++ interpreter aimed at processing C/C++ scripts.*

CINT is a part of ROOT. ROOT has a fascinating feature set, and it would be interesting to integrate as well. I would have done so except that ROOT is covered by the LGPL, and being a commercial developer myself, this license would never allow me to use the work in an actual project. Scintilla and CINT are covered by more commercial friendly licenses. I would definitely like to update this project in the future to integrate ROOT as well.

I downloaded and extracted CINT into the subfolder *cint* in our project folder. Unfortunately, the VC project files for CINT are actually wrappers around a **MAK** file. Because of this, I chose to just create another project and add the files that I needed. I created the project files in the subfolder *./cint/libcint*. This project natively creates static libraries, so there is nothing else to do here but compile.

Upon compiling, I ran into a small snag. Two functions, `G__SetGlobalcomp()` and `G__ForceBytecodecompilation()`, are defined in both *Method.cxx* and *v6_dmystrm.c*, and both files are needed to compile CINT. The *v6_dmystrm.c* versions are the ones we want, so I surrounded the two functions in *Method.cxx* with `#if 0`. I also had to do the same in *Apiifold.cxx*. I'm sure this is probably an oversight since the project is primarily for UNIX. Hopefully, this will be resolved in a later version.

Despite this minor hiccup, things are now compiling nicely. Let's add it to our project by added the following includes to the the *Stdafx.h* file...

Collapse | Copy Code

```
// CINT
#include "cint/G__ci.h"
#include "cint/src/Global.h"
```

We need to link as well, so we add the following lines to the *Stdafx.cpp* file...

Collapse | Copy Code

```
#ifdef _DEBUG
#   pragma comment( lib, "cint/libcint/Debug/libcint.lib" )
#else
#   pragma comment( lib, "cint/libcint/Release/libcint.lib" )
#endif
```

Creating the Scintilla Editor and Output windows

Now that both libraries are linked in, we can start the fun part of actually using them. To start, let's create the Scintilla editor by calling the function `InitialiseEditor()` from `OnInitDialog()`. We require a list of C/C++ keywords and a color scheme to complete the initialization. I used a color scheme I like: the original **Twilight** scheme from the old Borland compiler for DOS from which I learned C++ many moons ago. It should be straightforward to modify the colors to your own taste.

Collapse | Copy Code

```
// C++ keywords
static const char g_cppKeyWords[] =

    // Standard
    "asm auto bool break case catch char class const "
    "const_cast continue default delete do double "
    "dynamic_cast else enum explicit extern false finally "
    "float for friend goto if inline int long mutable "
    "namespace new operator private protected public "
    "register reinterpret_cast register return short signed "
    "sizeof static static_cast struct switch template "
    "this throw true try typedef typeid typename "
    "union unsigned using virtual void volatile "
    "wchar_t while "

    // Extended
    "__asm __asume __based __box __cdecl __declspec "
    "__delegate delegate depreciated dllexport dllimport "
    "event __event __except __fastcall __finally __forceinline "
    "__int8 __int16 __int32 __int64 __int128 __interface "
    "interface __leave naked noinline __noop noreturn "
    "nothrow novtable nullptr safecast __stdcall "
    "__try __except __finally __unaligned uuid __uuidof "
    "__virtual_inheritance";


/// Scintilla Colors structure
struct SScintillaColors
{   int         iItem;
    COLORREF    rgb;
};

// A few basic colors
const COLORREF black = RGB( 0, 0, 0 );
const COLORREF white = RGB( 255, 255, 255 );
const COLORREF green = RGB( 0, 255, 0 );
const COLORREF red = RGB( 255, 0, 0 );
const COLORREF blue = RGB( 0, 0, 255 );
const COLORREF yellow = RGB( 255, 255, 0 );
const COLORREF magenta = RGB( 255, 0, 255 );
const COLORREF cyan = RGB( 0, 255, 255 );

/// Default color scheme
static SScintillaColors g_rgbSyntaxCpp[] =
{
    {   SCE_C_COMMENT,          green },
    {   SCE_C_COMMENTLINE,      green },
    {   SCE_C_COMMENTDOC,       green },
    {   SCE_C_NUMBER,           magenta },
    {   SCE_C_STRING,           yellow },
    {   SCE_C_CHARACTER,        yellow },
    {   SCE_C_UUID,             cyan },
    {   SCE_C_OPERATOR,         red },
    {   SCE_C_PREPROCESSOR,     cyan },
    {   SCE_C_WORD,             cyan },
    {   -1,                     0 }
};

void CCintDlg::InitialiseEditor()
{
    // Punt if we already have a window
    if ( ::IsWindow( m_hwndEditor ) ) return;

    // Create editor window
    m_hwndEditor = CreateWindowEx(  0, "Scintilla", "",
                                    WS_CHILD | WS_VISIBLE | WS_TABSTOP |
                                    WS_CLIPCHILDREN,
                                    10, 10, 500, 400,
                                    GetSafeHwnd(), NULL /*(HMENU)GuiID*/,
                                    AfxGetApp()->m_hInstance, NULL );

    // Did we get the editor window?
    if ( !::IsWindow( m_hwndEditor ) )
    {   TRACE( "Unable to create editor window\n" );
        return;
    } // end if

    // CPP lexer
    SendEditor( SCI_SETLEXER, SCLEX_CPP );

    // Set number of style bits to use
    SendEditor( SCI_SETSTYLEBITS, 5 );

    // Set tab width
    SendEditor( SCI_SETTABWIDTH, 4 );

    // Use CPP keywords
    SendEditor( SCI_SETKEYWORDS, 0, (LPARAM)g_cppKeyWords );

    // Set up the global default style. These attributes
    // are used wherever no explicit choices are made.
    SetAStyle( STYLE_DEFAULT, white, black, 10, "Courier New" );

    // Set caret foreground color
    SendEditor( SCI_SETCARETFORE, RGB( 255, 255, 255 ) );

    // Set all styles
    SendEditor( SCI_STYLECLEARALL );
```

I also used Scintilla for the output window. Just to show a different method. I accessed the output window more directly using `::SendMessage()`.

Collapse | Copy Code

```
void CCintDlg::InitialiseOutput()
{
    // Punt if we already have a window
    if ( ::IsWindow( m_hwndOutput ) ) return;

    // Create editor window
    m_hwndOutput = CreateWindowEx(  0, "Scintilla", "",
                                    WS_CHILD | WS_VISIBLE | WS_TABSTOP |
                                    WS_CLIPCHILDREN,
                                    10, 10, 500, 400,
                                    GetSafeHwnd(), NULL /*(HMENU)GuiID*/,
                                    AfxGetApp()->m_hInstance, NULL );

    // Did we get the editor window?
    if ( !::IsWindow( m_hwndEditor ) )
    {   TRACE( "Unable to create editor window\n" );
        return;
    } // end if

    // Set number of style bits to use
    ::SendMessage( m_hwndOutput, SCI_SETSTYLEBITS, 5, 0L );

    // Set tab width
    ::SendMessage( m_hwndOutput, SCI_SETTABWIDTH, 4, 0L );

    // Set foreground color
    ::SendMessage( m_hwndOutput, SCI_STYLESETFORE,
                   STYLE_DEFAULT, (LPARAM)RGB( 255, 255, 255 ) );

    // Set background color
    ::SendMessage( m_hwndOutput, SCI_STYLESETBACK, STYLE_DEFAULT, (LPARAM)RGB( 0, 0, 0 ) );

    // Set font
    ::SendMessage( m_hwndOutput, SCI_STYLESETFONT, STYLE_DEFAULT, (LPARAM)"Courier New" );

    // Set selection color
    ::SendMessage( m_hwndOutput, SCI_SETSELBACK, (WPARAM)TRUE, (LPARAM)RGB( 0, 0, 255 ) );

    // Set all styles
    ::SendMessage( m_hwndOutput, SCI_STYLECLEARALL, 0, 0L );
}
```

To position the windows, I used the following code. It's straightforward as we are just dealing with plain old window handles...

Collapse | Copy Code

```
BOOL CCintDlg::Size()
{
    // Ensure valid window
    if ( !::IsWindow( GetSafeHwnd() ) )
        return FALSE;

    // Get window size
    RECT rect, ctrl;
    GetClientRect( &rect );
    CopyRect( &ctrl, &rect );

    // Position the editor window
    ctrl.bottom -= ( 6 * 24 );
    CWnd *pWnd = CWnd::FromHandle( m_hwndEditor );
    if ( pWnd ) pWnd->MoveWindow( &ctrl );

    // Position the output window
    ctrl.top = ctrl.bottom;
    ctrl.bottom = rect.bottom;
    pWnd = CWnd::FromHandle( m_hwndOutput );
    if ( pWnd ) pWnd->MoveWindow( &ctrl );

    return TRUE;
}
```

Executing the code with CINT

Of course, we want to be able to execute the script, and have any output generated accessible in a cut-and-pasteable window. CINT sends the output from the script to the standard output stream STDOUT. We need to intercept this data. So to make this simple, I appropriated the class `CHookStdio` from the Open Source project rulib (don't worry, no license violation here). `CHookStdio` allows us to easily hook STDOUT and access the data. Note that we have to pass a parameter to `CHookStdio` indicating how much buffer space we need. Be aware of this size when writing to your scripts. I set this to 64K.

So the steps to execute our script are now...

- Get the text from Scintilla
- Hook STDOUT
- Send to CINT for processing
- Check for CINT errors

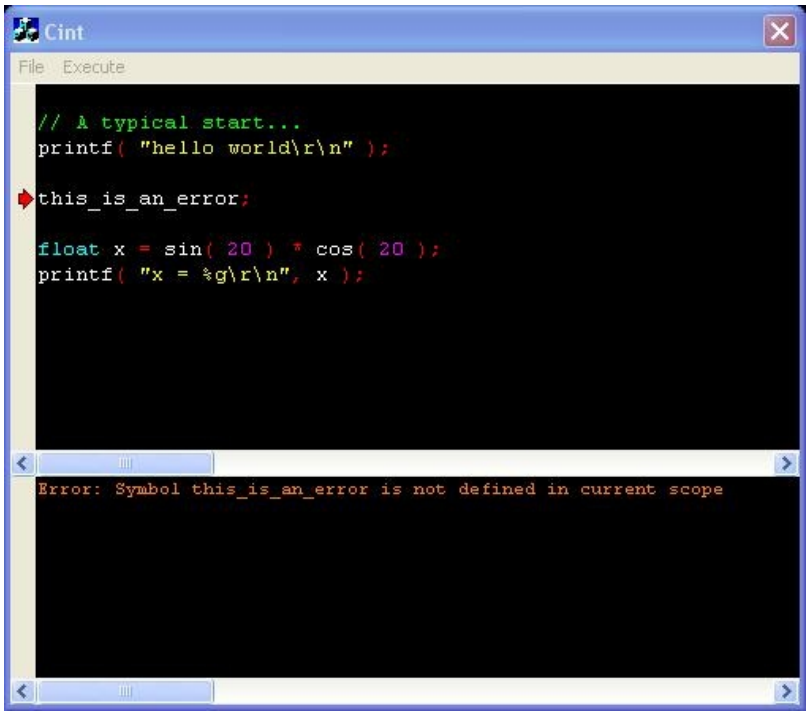
- Write the hooked STDOUT data to the output window

And here are the details.

Collapse | Copy Code

```
void CCintDlg::OnExecute()
{
    // Reset CINT
    G__scratch_all();
    g_sCintLastError = "";

    // Reset Scintilla
    SendEditor( SCI_MARKERDELETEALL, 0 );

    // Clear output window
    ::SendMessage( m_hwndOutput, SCI_SETTEXT, 0, (WPARAM)"" );

    // Get the editor window handle
    CWnd *pWnd = CWnd::FromHandle( m_hwndEditor );
    if ( !pWnd ) return;

    // Get the script text
    CString strScript;
    pWnd->GetWindowText( strScript );
    if ( strScript.IsEmpty() ) return;

    // Must add to get proper line number
    strScript = "#line 0\r\n" + strScript;

    // Hook stdio output
    CHookStdio hs( STD_OUTPUT_HANDLE );

    // Set error callback function
    G__set_errmsgcallback( &CCintDlg::CintError );

    // Execute the program
    if ( !G__int( G__exec_text( (LPCTSTR)strScript ) ) )
    {
        // Initilaize error markers
        SendEditor( SCI_MARKERDEFINE, 0, SC_MARK_SHORTARROW );
        SendEditor( SCI_MARKERSETFORE, 0, RGB( 80, 0, 0 ) );
        SendEditor( SCI_MARKERSETBACK, 0, RGB( 255, 0, 0 ) );

        // Set error marker to proper line
        int nErrLine = G__lasterror_linenum();
        SendEditor( SCI_MARKERADD, nErrLine - 1, 0 );

        // Show the error string
        ShowError( g_sCintLastError.c_str() );

        return;

    } // end if

    // Set foreground color
    ::SendMessage( m_hwndOutput, SCI_STYLESETFORE,
                   STYLE_DEFAULT, (LPARAM)RGB( 255, 255, 255 ) );
    ::SendMessage( m_hwndOutput, SCI_STYLECLEARALL, 0, 0L );

    // Get output
    char buf[ 64 * 1024 ] = "";
    buf[ hs.Read( buf, sizeof( buf ) - 1 ) ] = 0;

    // Show script output
    if ( *buf ) ::SendMessage( m_hwndOutput, SCI_SETTEXT, 0, (WPARAM)buf );
}
```

This function also includes the code for highlighting errors in the script. Here is a screenshot of what an error looks like.



Conclusion

That sums it up. So have fun.

One of the many practical uses I can think of for this project is generating tedious lookup tables which often come up when optimizing code. Now I can create a simple script and run it when I needed, instead of wasting time with an entire project just to generate a simple table. (Although I have to admit to using PHP for this lately.)

Here is a script to generate a square root lookup table...

Collapse | Copy Code

```
// Start the table
printf( "const double g_rdLookup_Sqrt[] = \r\n{\r\n\t" );

// Generate items
for ( int i = 0; i < 100; i++ )
{
    // Add value separator
    if ( i ) printf( ", " );

    // Break into reasonable pieces
    if ( i && !( i % 8 ) ) printf( "\r\n\t" );

    // Calculate value
    printf( "%g", (double)sqrt( (double)i ) );
}

// Complete the table
printf( "\r\n};" );
```

And here's the output...

Collapse | Copy Code

```
const double g_rdLookup_Sqrt[] =
{
    0, 1, 1.41421, 1.73205, 2, 2.23607, 2.44949, 2.64575,
    2.82843, 3, 3.16228, 3.31662, 3.4641, 3.60555, 3.74166, 3.87298,
    4, 4.12311, 4.24264, 4.3589, 4.47214, 4.58258, 4.69042, 4.79583,
    4.89898, 5, 5.09902, 5.19615, 5.2915, 5.38516, 5.47723, 5.56776,
    5.65685, 5.74456, 5.83095, 5.91608, 6, 6.08276, 6.16441, 6.245,
    6.32456, 6.40312, 6.48074, 6.55744, 6.63325, 6.7082, 6.78233, 6.85565,
    6.9282, 7, 7.07107, 7.14143, 7.2111, 7.28011, 7.34847, 7.4162,
    7.48331, 7.54983, 7.61577, 7.68115, 7.74597, 7.81025, 7.87401, 7.93725,
    8, 8.06226, 8.12404, 8.18535, 8.24621, 8.30662, 8.3666, 8.42615,
    8.48528, 8.544, 8.60233, 8.66025, 8.7178, 8.77496, 8.83176, 8.88819,
    8.94427, 9, 9.05539, 9.11043, 9.16515, 9.21954, 9.27362, 9.32738,
    9.38083, 9.43398, 9.48683, 9.53939, 9.59166, 9.64365, 9.69536, 9.74679,
    9.79796, 9.84886, 9.89949, 9.94987
};
```

License

About the Author

Robert Umbehant

Web Developer

United States

Member

Article Top

Sign Up to vote for this article

Comments and Discussions

You must Sign In to use this message board.

FAQ Search

Noise Tolerance: Medium | Layout: Normal | Per page: 25 | Update

Msgs 1 to 19 of 19 (Total in Forum: 19) (Refresh) First Prev Next

How can I declare functions?	**im_chc**	**8:18 6 Sep '10**
for, while do while switch instrucction doesn't work	**malfaro**	**11:08 30 Jan '08**
Re: for, while do while switch instrucction doesn't work	malfaro	14:05 2 Feb '08
Compiling Cint	**WalderMort**	**16:13 19 Aug '07**
Running code from a dll	**Elm Costa**	**2:16 4 Dec '06**
CINT Callbacks	**jkhax0r**	**14:10 9 Nov '06**
Runtime errors using VC 7.0 - any ideas ?	**Michael B Pliam**	**14:13 13 Aug '06**
Re: Runtime errors using VC 7.0 - any ideas ?	Robert Umbehant	5:13 15 Aug '06
Re: Runtime errors using VC 7.0 - any ideas ? [modified]	Michael B Pliam	9:32 16 Aug '06
ROOT license - LGPL	**Matthew Hannigan**	**16:02 12 Jul '06**
Re: ROOT license - LGPL [modified]	Robert Umbehant	16:38 12 Jul '06
AngelScript	**Gilad Novik**	**18:15 10 Jul '06**
UNICODE [modified]	**John A. Johnson**	**4:14 10 Jul '06**
Re: UNICODE	Robert Umbehant	9:50 10 Jul '06
Other C Interpreters.	**Neville Franks**	**12:22 9 Jul '06**
Re: Other C Interpreters.	Robert Umbehant	15:17 9 Jul '06
Re: Other C Interpreters.	cprojectuser	23:18 8 Aug '06
Neat!	**Jerry Evans**	**3:27 9 Jul '06**
Re: Neat!	Robert Umbehant	7:52 9 Jul '06

Last Visit: 19:48 30 Jan '11 Last Update: 20:04 30 Jan '11 1

General News Question Answer Joke Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+PgUp/PgDown to switch pages.

See Also...

Using Scintilla for syntax coloring in MFC
The article describes how to use the scintilla...

Embeddable script editor for MFC applications
A library that allows you to embed scripting...

CUltraPadWnd - A Simple Syntax Coloring Control Based on the MFC CWnd Class
An article on a simple syntax coloring control

2D LUA Based Robot Simulator
An article on designing your own robot simulator

Using AvalonEdit (WPF Text Editor)
AvalonEdit is an extensible Open-Source text...

Announcements

Build Azure App - Win a Laptop

The Daily Insider

30 free programming books
Daily News: Signup now.