CEFPython

Developing applications using Chromium

Jason Jones
@JasonProgrammer

simpletutorials.com/cef
What is Chromium?

The code behind Google’s Chrome browser.

*Google has Chromium OS, but we’re talking about the browser

[http://www.chromium.org/Home](http://www.chromium.org/Home)
What is CEF?
Chromium Embedded Framework

C/C++ wrapper around Chromium.

Helps embed Chromium into applications.
What is CEF?

What is CEFPython?
...a Python wrapper for CEF.

CEFPython

CEF

Chromium
Why use it?

- You’re familiar with web design.
- You want a responsive, cross-platform UI.
- You want custom UI controls.
- You want to provide a client with a custom browser.
What’s something I’d like to do with it?
Desktop and Mobile HTML5 game framework
A fast, free and fun open source framework for Canvas and WebGL powered browser games.

DOWNLOAD & GET STARTED
Download or Fork via Github

2.3.0
Who uses CEF?
Note the responsive design (resize-friendly)
Applications using CEF [edit]

- Adobe Brackets - open source code editor for the web
- Adobe Edge Animate - multimedia authoring tool
- Adobe Edge Reflow - responsive web design tool
- Adobe Dreamweaver CC - web design software
- Alpha Anywhere - web application development tool
- Amazon Music - media player for PC and Mac
- AppJS - build desktop applications using Node.js[21]
- AOL Instant Messenger - instant messaging client that uses CEF on Windows
- Battle.net Client -
- Chromium Tab - another Chromium inside Chrome Chromium Tab extension[9]
- Cubiclez - integrates content into the desktop[22]
- Desura - online game platform
- Dish World IPTV - streaming video platform
- Evernote - notetaking software
- Eve Launcher - launcher for Eve Online on Mac OS X[23] beta launcher on Windows
- EexeOutput - allows deploying HTML5/PHP applications as desktop applications[24]
- ExpanDrive - network file system client
- Facebook Messenger for Windows
- fan - movie compilation and playback app for Windows[25]
- Free Download Manager 5.0 Alpha - free downloads manager (alpha version)
- JMS4 - program for joinery companies
- GitHub for Windows - GitHub client software for Windows
- Google Web Designer - create interactive HTML5 sites and ads
- Inky - smart email app
- Intel AppUp Encapsulator - Intel app store software
- Janetter - Twitter client software[26]
- Key4Two - hardware-level secured sharing network[27]
- Mailbird - Windows email software
- mChef - miRC browser plugin[28]
- MedaMan - organizer software
- MetaVR - geographic simulation software[29]
- MTG Studio - game organizer software[30]
- OpenSpace3D - 3D development platform[31]
- PHP Desktop - develop desktop applications using PHP, HTML 5, Javascript and SQLite[32]
- PokerStars - the largest online poker cardroom in the world
- Qobuz Desktop - HD download and streaming music platform[33]
- Quantel Webtools - Webtools allows third party web applications to appear within Quantel editors
- Rdio - streaming music platform
- Spotify - streaming music platform
- MEC Music - streaming music application[34]
- Steam Client - online game platform
- Telebreeze player - multipurpose streaming video player[35]
- Tencent QQ - instant messaging program (its QPlus part) and web browser
- Trend Micro - internet security software
- Uplay - online game platform
- WSEAB Desktop - allows deploying HTML5 applications as desktop applications[36]
- yasoon - app store & platform for Microsoft Outlook
- Macaw Web Editor - commercial WYSIWYG editor used for web development

CEFPython Requirements

Windows XP SP3+
Python 2.7 (no Python 3 packages yet)

http://www.magnpcss.net/cef_downloads/
What UI frameworks work with CEFPython?
Quite a few...

Kivy - MIT License
PyWin32 - Python Software Foundation license
Panda3D - Modified BSD - 3D game engine (Disney)
wxPython - based on wxWidgets C++ library
PyGTK - LGPL
PySide - LGPL

PyQt - GPL, Fee for commercial use

*No CEFPython examples provided for Tkinter
I chose **wxPython**.

- Open source
- Free for commercial use
- Cross-platform
- Based on wxWidgets (C++ library)
A great wxPython example comes with cefpython.
Use the mouse context menu to go Back/Forward in history navigation.

Table of contents

1. Google search
2. User agent
3. Popups
4. HTML 5 video
5. Developer Tools
6. Downloads
7. HTML controls
8. Browser object
9. Frame object
10. Javascript bindings
11. Javascript callbacks
12. Python callbacks
13. Display handler
14. Keyboard handler
15. Request handler
16. Cookie tests
17. Load handler
18. Javascript Dialog handler
19. Other tests

Google search
http://www.google.com/

User agent
So does everyone who runs my application need to install Python?
We can create an executable from Python code.

http://www.py2exe.org/
from distutils.core import setup
import py2exe
from cefpython3 import cefpython
import wx
import time
import re
import uuid
import platform
import inspect
import struct
import urllib
setup(
    windows=[{
        'script': 'wxpython.py'
    }]
)
set **PYTHONPATH**=C:\Python27;C:\Python27\Lib;C:\Python27\DLLs
C:\Python27\python.exe **build_exe.py** py2exe

wxpython.exe

(no Python interpreter needed to run this)
Use the mouse context menu to go Back/Forward in history navigation.

Table of contents

1. Google search
2. User agent
3. Popups
4. HTML 5 video
5. Developer Tools
6. Downloads
7. HTML controls
8. Browser object
9. Frame object
10. Javascript bindings
11. Javascript callbacks
12. Python callbacks
13. Display handler
14. Keyboard handler
15. Request handler
What does task manager look like?
<table>
<thead>
<tr>
<th>Image Name</th>
<th>User Name</th>
<th>CPU</th>
<th>Memory (Private Working Set)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>soffice.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>152 K</td>
<td>OpenOffice.org 3.3</td>
</tr>
<tr>
<td>stray64.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>1,096 K</td>
<td>IDT PC Audio</td>
</tr>
<tr>
<td>subprocess.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>8,800 K</td>
<td>subprocess.exe</td>
</tr>
<tr>
<td>subprocess.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>26,392 K</td>
<td>subprocess.exe</td>
</tr>
<tr>
<td>taskhost.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>3,196 K</td>
<td>Host Process for Windows Tasks</td>
</tr>
<tr>
<td>taskhost.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>432 K</td>
<td>Host Process for Windows Tasks</td>
</tr>
<tr>
<td>taskmgr.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>5,944 K</td>
<td>Windows Task Manager</td>
</tr>
<tr>
<td>unsecapp.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>736 K</td>
<td>Sink to receive asynchronous call...</td>
</tr>
<tr>
<td>upeksrvr.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>1,884 K</td>
<td></td>
</tr>
<tr>
<td>VBoxSVC.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>3,300 K</td>
<td>VirtualBox Interface</td>
</tr>
<tr>
<td>vcpkgsvr.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>76,680 K</td>
<td>Microsoft (R) Visual C++ Package...</td>
</tr>
<tr>
<td>vcpkgsvr.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>75,056 K</td>
<td>Microsoft (R) Visual C++ Package...</td>
</tr>
<tr>
<td>vcpkgsvr.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>26,480 K</td>
<td>Microsoft (R) Visual C++ Package...</td>
</tr>
<tr>
<td>VirtualBox.exe</td>
<td>JasonJones</td>
<td>07</td>
<td>60,520 K</td>
<td>Oracle VM VirtualBox Manager</td>
</tr>
<tr>
<td>VirtualBox.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>13,832 K</td>
<td>Oracle VM VirtualBox Manager</td>
</tr>
<tr>
<td>vpngui.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>1,540 K</td>
<td>Cisco AnyConnect User Interface</td>
</tr>
<tr>
<td>winlogon.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>812 K</td>
<td></td>
</tr>
<tr>
<td>wuaupd.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>688 K</td>
<td>Windows Update</td>
</tr>
<tr>
<td>wxpython.exe</td>
<td>JasonJones</td>
<td>00</td>
<td>72,172 K</td>
<td>wxpython.exe</td>
</tr>
</tbody>
</table>

Processes: 158  CPU Usage: 8%  Physical Memory: 86%
Multiple Processes

Chromium uses multiple processes and talks between those processes (IPC).
What processes are involved?
When debugging, keep in mind there are multiple processes.

If your breakpoint isn’t being hit, you might need to use a `__debugbreak()`.
A separate (lightweight) process to be used as each render process.
Chromium uses multiple threads. Each thread has a message loop.

<table>
<thead>
<tr>
<th>User Interface</th>
<th>Safe Browsing</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO</td>
<td>History</td>
</tr>
<tr>
<td>File</td>
<td>Proxy</td>
</tr>
<tr>
<td>DB</td>
<td></td>
</tr>
</tbody>
</table>

http://www.chromium.org/developers/design-documents/threading
To run the message loop...

Call a *blocking* function that runs the message loops

-OR-

Call `DoMessageLoopWork` on a timer or on an idle event.
Don’t starve the message loop.
Sandbox

The render process can only access:

- CPU
- Memory

http://dev.chromium.org/developers/design-documents/sandbox/Sandbox-FAQ
A simple application to search the 50 states by name.
CherryPy
A Minimalist Python Web Framework

CHerrPy IS AS EASY AS...

```python
import cherrypy

class HelloWorld(object):
    def index(self):
        return "Hello World!"
    index.exposed = True

cherrypy.quickstart(HelloWorld())
```

Multithreaded, production ready web server and framework (WSGI)
import json
import cherrypy
from cherrypy import tools

def error_page_404(status, message, traceback, version):
    return "404 Error!"

class HomeController():
    @cherrypy.expose
    @cherrypy.tools.json_out()
    def GetStates(self):
        with open('states.json', 'r') as chat_file:
            return json.loads(chat_file.read())

def start_server():
    cherrypy.tree.mount(HomeController(), '/')
    cherrypy.config.update({'error_page.404': error_page_404})
    cherrypy.config.update({'server.socket_port': 9090})
    cherrypy.engine.start()

if __name__ == '__main__':
    start_server()
states.json

[
  "Alabama",
  "Alaska",
  "Arizona",
  "Arkansas",
  "California",
  "Colorado",
  "Connecticut",
  "Delaware",
  "District Of Columbia",
  "Florida",
  "Georgia",
  ...(more)...
Quickstart

Eager to get started? This page gives a good introduction in how to get started with Requests.

First, make sure that:

- Requests is [installed](https://requests.readthedocs.io/en/master/)
- Requests is [up-to-date](https://pypi.org/project/requests/)

Let’s get started with some simple examples.

Make a Request

Making a request with Requests is very simple.

Begin by importing the Requests module:

```python
>>> import requests
```

Now, let’s try to get a webpage. For this example, let’s get GitHub’s public timeline

```python
>>> r = requests.get('https://api.github.com/events')
```
What about Python 3?

CEF Python

Introduction

CEF Python is an open source project founded by Czarek Tomczak in 2012 to provide python bindings for the Chromium Embedded Framework. See the growing list of applications using CEF on wikipedia. Examples of embedding CEF browser are available for many popular GUI toolkits including: wxPython, PyGTK, PyQt, PySide, Kivy, Panda3D and PyWin32.

Some use cases for CEF:

- Embed a web browser control with great HTML5 support (based on Chromium)
- Use it to create a HTML5 based GUI in an application. This can act as a replacement for GUI toolkits like wxWidgets/Qt/Gtk. For native communication between javascript and python use javascript bindings. Another option is to run an internal python web server and use websockets/XMLHttpRequest for js<>python communication. This way you can write a desktop app in the same way you write web apps.
- Render web content off-screen in applications that use custom drawing frameworks. See the Kivy and Panda3D examples.
- Use it for automated testing of existing web applications. Use it for web scraping, or as a web crawler or other kind of internet bots.

Supported Python versions and platforms

- Supported Python versions: 2.7 (Python 3.4 will be supported soon, see Issue 121)
- Supported platforms: Windows, Linux, Mac (both 32bit and 64bit binaries are available for all platforms)
Issue 121: Support for Python 3.4

18 people starred this issue and may be notified of changes.

Status: Accepted
Owner: czarek.t...@gmail.com
Type: Defect
Priority: Medium
Blocked on: issue 110

Project Member
Reported by czarek.t...@gmail.com, Apr 8, 2014

The last Py3 binaries available are for Python 3.2 which is outdated.

Project Member
#5 czarek.t...@gmail.com

Jul 31, 2014

To compile with Python 3.4 it is required to update Cython to version 0.20 (or later) which added support for CPython 3.4 [1]. Issue 110 needs to be fixed first.


Blocked on: cefpython:110

Project Member
#8 czarek.t...@gmail.com

Jan 17, 2015

In Makefiles there are defined includes like -I/usr/include/Python2.7. To enable compilation for both 2.7 and 3.4 we will have to introduce flags like --py27 --py34 which will set appropriate include paths based on Python version.

Another issue reported by Alex is in compile.py script: raw_input() is no more available in Python 3, use input() instead.

Project Member
#9 mese1...@gmail.com

May 3 (4 days ago)

Is there any update about python 3?
I hope we can have it soon.
Thanks.
What is Cython?

Python + C data types

It’s a language (superset of Python) that compiles to C++. It’s used for optimization and interfacing to C/C++ code.
Can I update CEF?

What if I want the latest version of CEF/Chromium?
CEF Builds

Latest nightly - Rev. 2378
Requires Visual Studio 2013

https://cefbuilds.com/
Python extensions for Windows need to be built with the same version of VC++ used to build Python.
4.1. A Cookbook Approach

There are two approaches to building extension modules on Windows, just as there are on Unix: use the `distutils` package to control the build process, or do things manually. The `distutils` approach works well for most extensions; documentation on using `distutils` to build and package extension modules is available in *Distributing Python Modules (Legacy version)*. This section describes the manual approach to building Python extensions written in C or C++.

To build extensions using these instructions, you need to have a copy of the Python sources of the same version as your installed Python. You will need Microsoft Visual C++ “Developer Studio”; project files are supplied for VC++ version 7.1, but you can use older versions of VC++. Notice that you should use the same version of VC++ that was used to build Python itself. The example files described here are distributed with the Python sources in the `PC\example_nt` directory.
<table>
<thead>
<tr>
<th>Python version</th>
<th>VC++ version</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5.6</td>
<td>7.1</td>
</tr>
<tr>
<td>2.6.9</td>
<td>9.0</td>
</tr>
<tr>
<td>2.7.6</td>
<td>9.0</td>
</tr>
<tr>
<td>3.2.3</td>
<td>9.0</td>
</tr>
<tr>
<td>3.3.5</td>
<td>10.0</td>
</tr>
<tr>
<td>3.4.0</td>
<td>10.0</td>
</tr>
</tbody>
</table>
Visual C++ for VS2013 is version 18

Are there Python binaries built with VS2013?
Building Python 3.3.4 with Visual Studio 2013

4th revision, April 12, 2014.
A document history can be found at the very end of this page

Downloads
For the impatient:

- Python 3.3.4 32-bit build using Visual Studio 2013.1
- Sources used for Python 3.3.4 64-bit build using Visual Studio 2013.1
- Python 3.3.4 64-bit build using Visual Studio 2013.1
- Sources used for Python 3.3.4 64-bit build using Visual Studio 2013.1

*Note*: these builds use updated versions of OpenSSL (1.0.1g) - fix for the heartbleed issue - and SQLite (3.8.4.3), plus tons of other new goodies. Plus document revision 4 adds a new
Is it possible to use Python 3 and the CEF Nightly build?
Yes. You can write Python-C++ bindings with SWIG.
Simplified Wrapper and Interface Generator

http://en.wikipedia.org/wiki/SWIG
http://www.swig.org/tutorial.html
SWIG can generate wrapper code for 23 languages:

- Allegro CL
- C#
- CFFI
- CLISP
- Chicken
- D
- Go
- Guile
- Java
- Javascript
- Lua
- Modula-3
- Mzscheme
- OCAML
- Octave
- Perl
- PHP
- Python
- R
- Ruby
- Scilab
- Tcl
- UFFI
Install SWIG

Download
Extract
Add directory to your PATH
Install VS

Visual Studio Community 2013

A full-featured IDE – Free for students, open source contributors, and small teams. Start coding the app of your dreams for Windows, Android, and iOS.
Interface

File

Helps SWIG know what to wrap

/* CefSwigApp.i */

module cefswig

{%
#include "CefSwigApp.h"
%
}

class CefSwigApp
{
public:
    CefSwigApp();
    int CreateBrowser();
    void DoMessageLoopWork();
};
import cefswig

Our Python module:

C++ wrapper we will build with our source to produce a pyd (DLL) file
CEF C++ Solution
<table>
<thead>
<tr>
<th><strong>Project Defaults</strong></th>
<th><strong>Configuration Type</strong></th>
<th><strong>Dynamic Library (.dll)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Use of MFC</td>
<td>Use Standard Windows Libraries</td>
<td></td>
</tr>
<tr>
<td>Character Set</td>
<td>Use Unicode Character Set</td>
<td></td>
</tr>
<tr>
<td>Common Language Runtime Support</td>
<td>No Common Language Runtime Support</td>
<td></td>
</tr>
<tr>
<td>Whole Program Optimization</td>
<td>No Whole Program Optimization</td>
<td></td>
</tr>
<tr>
<td>Windows Store App Support</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

**Output Directory**

Specifies a relative path to the output file directory; can include environment variables.
executable Directories
Path to use when searching for executable files while building a VC++ project. Corresponds to environment variable PATH.
Thanks!

Questions? @jasonprogrammer