# О проекте

Проект демонстрирует возможности использования браузера chromium в приложениях, написанных на C++, а также взаимодействие кода JavaScript и C++ в рамках библиотеки CEF (Chromium Embedded Browser).

CEF является кроссплатформенной библиотекой, но данный проект работает под Windows.

Функционал приложения: поиск в папке «Документы» файлов и папок по именам. Строка для поиска поддерживает специальные символы «?» и «\*». Интерфейс приложения выполнен в html-странице, а сам поиск — на C++. Результаты поиска записываются в таблицу на html-странице.

# Установка и настройка всего необходимого

## Скачивание CEF

Чтобы не собирать всю библиотеку вместе с chromium, можно скачать уже собранную с https://cefbuilds.com. В данном проекте используется версия CEF 3.2526.1364.gf6bf57b. Из архива распаковываем всё содержимое папки с номером версии в папку проекта ./thirdparty/cef.

## Сборка libcef\_dll\_wrapper

CEF предоставляет API для использования браузера на чистом C. Так как мы хотим использовать C++ API, то нам необходимо собрать обертку — статическую библиотеку, которая подключится к нашему приложению, и позволит использовать C++ API. Для этого нужно сгенерировать солюшен под MS Visual Studio и проекты с помощью CMake (version). CMake файлы CEF расположены в thirdparty/cef. Указываем эту же папку CMake'у и для генерирования проектов. Если есть желание указать другое расположение, то придется менять параметры и в проекте msvc. Нажимаем Configure. Параметры Cmake можно оставить по умолчанию. Нажимаем Generate. После генерирования открываем полученный солюшен и собираем проект libcef\_dll\_wrapper.

## Добавление ресурсов в debug версию

Для работы браузера необходимы файлы ресурсов, поэтому чтобы запускать отладочную версию из под студии, нужно скопировать содержимое папки ./thirdparty/cef/Resources в папку ./thirdparty/cef/Debug.

Теперь всё готово для сборки нашего приложения. Открываем солюшен ./build/windows/msvc/CefBindings.sln и собираем одноименный проект.

# Запуск

## Запуск версии debug

Проект msvc настроен для запуска из под отладчика. В свойствах проекта Debugging -> Command Arguments в параметре --url нужно задать полный путь к файлу index.html.

## Запуск версии release

Для версии release необходимо выполнить следующие шаги:

1. Поместить собранный файл CefBindings.exe из папки ./bin/windows/msvc/Win32/Release/ в папку ./thirdparty/cef/Release.
2. Скопировать содержимое папки ./thirdparty/cef/Resources в папку ./thirdparty/cef/Release.
3. Запустить CefBindings.exe url=”index.html”. Путь к файлу index.html необходимо указать полный, то есть с буквой диска.

# Обзор

## Подпроцессы

Важным моментом является то, что хромиум использует мультипроцессный режим ( подробнее здесь:<https://bitbucket.org/chromiumembedded/cef/wiki/Tutorial#markdown-header-architecture-in-60-seconds> ). Есть возможность запускать и в один процесс, но этот режим рекомендуется только для отладки. Основная логика приложения выполняется в процессе называемом «браузерным». Для рендеров, плагинов, аппаратного ускорения будут созданы отдельные подпроцессы. В данном проекте мы не создаем отдельных проектов для подпроцессов — запускается один и тот же CefBindings.exe, а параметрами командной строки назначается его роль. Вызов подпроцессов с параметрами выполняется хромиумом, нам достаточно только запустить основной процесс с указанием html-страницы. Это означает, что в данном проекте связь JS кода с C++ реализована в процессе рендера, а реализация поиска — в процессе браузера. Далее мы будем рассматривать только эти 2 процесса. Для взаимодействия между этими процессами используются механизм сообщений CEF. В методе main выполняется проверка, с какой ролью запущен процесс:

int APIENTRY \_tWinMain( \_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPTSTR lpCmdLine,

\_In\_ int nCmdShow )

{

CefMainArgs main\_args( hInstance );

// Optional implementation of the CefApp interface.

CefRefPtr<MyApp> app( new MyApp );

// Execute the sub-process logic, if any. This will either return immediately for the browser

// process or block until the sub-process should exit.

int exit\_code = CefExecuteProcess( main\_args, app.get(), nullptr );

if( exit\_code >= 0 )

{

// The sub-process terminated, exit now.

return exit\_code;

}

// continue for browser process

// ...

## Потоки

В нашем приложении мы имеем дело с двумя потоками: ui и renderer. Большинство методов требует выполнения в определенном потоке, поэтому в начале тела метода выполняется проверка текущего потока:

CEF\_REQUIRE\_UI\_THREAD();

или

CEF\_REQUIRE\_RENDERER\_THREAD();

Поток ui выполняется в процессе браузера. По сути это message loop. В этом же потоке будет работать поиск в папках.

*Примечание. Если бы в нашем приложении использовался поток, который уже выступает в роли message loop, например, в случае использования библиотеки Qt с вызовом метода Qapplication::exec(), то CEF имеет возможность интегрироваться с таким потоком вместо создания еще одного message loop. Для этого ему нужно передать флаг multi\_threaded\_message\_loop, тогда отдельный поток всё равно будет создан, но он не будет обрабатывать сообщения.*

Поток renderer выполняется в процессе рендера. В нем выполняется движок WebKit и JavaScript. Всё взаимодействие JS<->C++ будет происходить здесь.

## Классы

В основном экземпляры наших классы будут обернуты в CefRefPtr для подсчета ссылок, поэтому в private секциях классов добавлен макрос:

IMPLEMENT\_REFCOUNTING( V8Handler );

### Класс MyApp

class MyApp :

public CefApp,

public CefBrowserProcessHandler,

public CefRenderProcessHandler

{

public:

//...

Экземпляр этого класса будет создан в обоих процессах, но нужно помнить, что обработчики, переопределенные в нем, будут вызываться в соответствующем этому обработчику процессе. Так как создание экземпляров некоторых членов класса MyApp выполняется в этих обработчиках, то созданный экземпляр члена в одном процессе не будет создан в другом.

### Класс ClientHandler

class ClientHandler :

public CefClient,

public CefLifeSpanHandler

{

public:

//…

Будет создан только в процессе браузера. Содержит обработчик для получения сообщений от процесса рендерера.

### Класс V8Handler

class V8Handler : public CefV8Handler

{

public:

//…

Будет создан только в процессе рендерера. Его переопределенный метод Execute будет вызываться при вызове из JS методов, реализованных в C++.

### Класс Searcher

Содержит реализацию поиска.

## Создание JS методов с реализацией C++

В обработчике MyApp::OnContextCreated, который вызовется при создании контекста V8, добавляем в контекст методы, которые будут вызываться из JS, но реализация в C++. Методы добавляются в контекст объекта window.

// cpp method from js

CefRefPtr<CefV8Value> method = CefV8Value::CreateFunction( "searchInDocs", m\_v8handler );

object->SetValue( "searchInDocs", method, V8\_PROPERTY\_ATTRIBUTE\_READONLY );

searchInDocs вызывается по нажатию кнопки поиска и будет содержать один параметр - ключевое слово для поиска.

// create JS methods for register methods to call from cpp

object->SetValue( "registerAddMethod",

CefV8Value::CreateFunction( "registerAddMethod", m\_v8handler ),

V8\_PROPERTY\_ATTRIBUTE\_NONE );

registerAddMethod вызовется один раз при загрузке html страницы. В качестве параметра передается объект JS-метод - это метод function addItem(name, size, path)

для добавления элемента в таблицу. Мы сохраняем этот метод в объекте CefRefPtr<CefV8Value> m\_addItemMethod, чтобы вызывать его каждый раз при нахождении файла или папки.

## Flow

1. Пользователь вводит ключевое слово для поиска и нажимает кнопку для поиска. В JS вызывается метод searchInDocs с ключевым словом в качестве параметра.
2. Обработчик V8Handler::Execute в процессе рендерера получает из параметра ключевое слово и отправляет сообщение процессу браузера js\_search\_button\_pressed с ключевым словом в качестве параметра.
3. Обработчик ClientHandler::OnProcessMessageReceived в процессе браузера получает сообщение и запускает поиск.
4. При каждом совпадении отправляется сообщение item\_found\_msg процессу рендерера с информацией о найденном элементе.
5. Обработчик MyApp::OnProcessMessageReceived в процессе рендерера получает сообщение и вызывает ранее сохраненный метод JS для добавления элемента в таблицу.