We are happy to announce the release of **Awesomium.NET 1.7.5b**. This is a **beta** release with many new ***experimental*** features and enhancements.

- Download the Awesomium 1.7.5b SDK (includes the **Awesomium.NET** binaries and samples)
- Awesomium.NET 1.7.5 API Reference
- Setting up on Windows
- Setting up on Mac OS X

Major New Features

With this release we are introducing many important new features and improvements. Some of them may require refactoring your code.

> Please read: **Important Changes in v1.7.5** for a list and presentation of important API changes in version 1.7.5 that require refectoring your code.

Here is a short presentation of some of these new features and improvements:

Optimizations

All throughout the **Awesomium.NET** project, a series of **code optimizations have been applied that significantly improve the overall performance of Awesomium.NET components**.

- Pages and resources are now loaded faster.
- Overall memory usage improved.
- Rendering and resizing in *offscreen* surfaces is performed faster, using less resources.
- Interaction with pages using JavaScript is significantly improved (also see *New Javascript Integration Features* below).
- Awesomium.NET now logs messages to the log file (see: `LogPath` and `LogLevel`). Exceptions and their stack trace are also logged.
- Performance of Awesomium.NET's *synchronization context* (for non-UI environments and cross-thread interaction) has been improved and more features are added.

New Javascript Integration Features

Awesomium.NET v1.7.5 adds new features and improvements to Awesomium's Javascript Integration API. New features improve performance, allow easier interaction with the page and of the page with the hosting application and make the code needed to interact using JavaScript shorter, simpler and safer.

This is only a short list of the new features:

- JavaScript-related event handlers and custom JavaScript method handlers, are now called in a **Javascript Execution Context (JEC). Any `JSObject` instances passed, acquired or created in a JEC, are automatically disposed upon exiting the method (handler)** associated with the execution context. You no longer need to explicitly dispose JSObjects in those methods (by wrapping code with `using` statements for example).
- **Handlers executed in an asynchronous JEC** (handlers of asynchronous JavaScript-related event or

asynchronous custom JavaScript methods), **have immediate access to essential JavaScript objects of the loaded page's current JavaScript environment (such as *`window`*, *`document`* or the generics of *`Object`*)**, through `Global`. Applications no longer need to perform additional synchronous calls to acquire these objects (using `ExecuteJavascriptWithResult` for example).

- Binding errors, JavaScript errors or **exceptions that occur in code executed in a JEC, are silently handled and propagated to the JavaScript console** (see: `ConsoleMessage`).
- Users can now derive `JSObject` (it is no longer *`sealed`*), to create custom local JSObjects to pass to the page. The new Javascript Integration API allows **Awesomium.NET** cast these objects back to their original subclass, when they are re-acquired from V8.
- JavaScript clients can now interact with the hosting application through the API of the new **Javascript Interoperation Framework (JIF). The framework provides methods to acquire information about the hosting application and the running views, monitor global (`WebCore`) or `IWebView` events, control views (even views different than the one hosting the page), control the hosting UI (if any) and send messages to the hosting application synchronously or asynchronously, passing data and JavaScript objects, even when sending messages synchronously**. The hosting application can control the features available to JavaScript code through new settings added to `WebPreferences`. JIF provides most of the API a web-page would need to communicate with the application. This way applications don't have to design this interface themselves and they avoid numerous synchronous calls to the child-process to create global JavaScript objects (using `CreateGlobalJavascriptObject`) and bind to custom JavaScript methods.
- **Dynamic Language Runtime (DLR)** support on `JSObject` has been significantly improved. **Users can now pass managed handlers directly as arguments to a dynamic JavaScript expression, to be used as callbacks**. In C#, the JavaScript objects provided to an asynchronous handler through `Global`, are already of type *`dynamic`*. The overall performance and reliability of DLR on `JSObject` has also been improved and basic DLR support has been added to `JSValue`.
- All dynamic expressions now return either `JSValue` or `JSObject` (and subclasses of it). Unary and binary operations support has been added to `JSValue` and many new casting operators and features are added to both `JSValue` and `JSObject` that simplify code.
- `JSFunction`, a custom subclass of `JSObject` has been added to the API. **JavaScript objects of type *`function`* acquired from the page, are wrapped as `JSFunction` and implicit casting from `JSValue` is available**.
- **`JSObject` is now enumerable**. You can iterate through the ECMAScript enumerable property names of a JavaScript object as you would in JavaScript.
- **JSObject's indexer now also returns members of type *`function`* (methods) of a remote JavaScript object**.
- `JSObject` now allows users to **specify or acquire the ECMAScript property descriptor of a JavaScript object's property** (see: `JSPropertyDescriptor`, `JSObject[String,...,JSPropertyDescriptor]` and `JSObject.GetPropertyDescriptor`).
- You can now **check for `NaN` and `Infitity` JavaScript values through `JSValue`**.
- An indexer has been added to `JSValue` that among other features, allows users **edit JavaScript arrays directly from `JSValue`**.
- **Awesomium.NET's new Javascript Integration fixes issues with Awesomium's DOM**. For example, **JavaScript *`navigator.language`* now correctly reports a value related to the hosting application's `CultureInfo`** and extension methods have been added to the `Utilities` of technology-specific assemblies that help you set the application's culture and thus also control the appearance of web applications that rely on *`navigator.language`*. Also, **the missing *`click`* method has been added to the prototype of `HTMLAnchorElement`** that allows JavaScript simulate a click to a link.

- **Touch events are now properly fired on JavaScript** when using the WPF `WebControl` with a multi-touch surface (or code the new `InjectTouchEvent` method with any web-view component). Event listener assignment properties for **touch events** have been added to `Node`, the base class of all DOM elements.
- Many more fixes and reliability improvements under the hood.

For more details and presentation of the revemped Javascript Integration API, read the following articles:

- Introduction to JavaScript Integration
- Synchronous & Asynchronous API
- Javascript Execution Context (JEC)
- Dynamic Language Runtime (DLR) Support
- Javascript Interoperation Framework (JIF)

New WPF Design-Time Support

Three new WPF Designer support assemblies have been added to the SDK that provide design-time features for **Awesomium.NET** WPF components for **Visual Studio 2010, 2012 and 2013**.

Here's a list of the new design-time support features provided to WPF components:

- You can now **edit the `WebPreferences` of a `WebSessionProvider` directly from the Properties window** when a `WebSessionProvider` is selected in XAML (even if a `WebPreferences` element is not added in XAML). Even if properties in the Properties window are not sorted by Category, **a dialog is available for editing `WebPreferences`**.
- **Default values of preferences are shown to the Properties window** or dialog and **tooltips with descriptions for each setting are available**.
- A dialog has been added that allows you to **see and edit the properties of `DataSources` of a `WebSessionProvider` when it's selected in XAML**.
- **New Properties window editors** have been added for most of the properties of a WPF `WebControl`.
- Awesomium.NET's designer of a `WebControl` now controls the settings and the availability of properties of a `WebControl` in the Properties window, based on other settings of the application or the `WebControl` itself.
- WebControl's editor of the `WebSession` property in the Properties window provides a drop-down menu that allows you to select from existing `WebSessionProvider` resources. Upon selection, **XAML is automatically generated**.
- When you drag and drop a `WebControl` from the Visual Studio toolbox in Visual Studio 2012 and 2013, the control automatically performs initialization and **fills the parent container in the designer**.

Install the SDK now and explore the new WPF design-time features.

As always, don't forget to **download the ClickOnce WPF demo, from here**.

PDF Files Viewer

Awesomium.NET v1.7.5 incorporates **PDF.js**. When you navigate to a PDF file, Awesomium.NET's **PDF.js** implementation is loaded and shows the PDF file in any Awesomium.NET web-view component.

> This technology is **experimental**.

- Users can still **download the original PDF file** using the **Download** button available in the viewer's toolbar.

- **Awesomium.NET** automatically replaces HTML `<object>` tags in a page that attempt to load Adobe's PDF Reader plugin (that is officially not supported by Awesomium), with an `<iframe>` that loads Awesomium.NET's **PDF.js** implementation.
- Users can **enable or disable PDF.js support** through `WebPreferences.PdfJS`.
- The **PDF viewer is automatically localized** based on the value of *`navigator.language`*, thus based on the application's current `CultureInfo` (see new Javascript Integration features above).

Run an Awesomium.NET sample or demo now, and navigate to the PDF version of this CHANGELOG.

Asynchronous ResourceInterceptor

A single powerful new API member, `IgnoreDataSources`, now allows users to **asynchronously load resources for remote pages**, combining the power of an `IResourceInterceptor` implementation and of a custom `DataSource`.

Members of an `IResourceInterceptor` are called in the I/O thread and providing a response must be performed synchronously (actually, delaying the I/O thread in any way can cause all sorts of side effects). However, custom DataSources (designed to load local pages and resources on a web-view), can provide responses and resources asynchronously. Using the available new API, users can now combine the power of these two classes to asynchronously load resources for remote pages:

Simple WebView Example:

1. Create a custom `DataSource` (or `AsyncDataSource` or use any of the predefined **Awesomium.NET** DataSources to load resources from assembly resources, a local directory or a compressed PAK file).
2. Implement `OnRequest` (or `LoadResourceAsync` respectively) to asynchronously provide one or more resources upon certain requests. (Skip this step if you use a predefined `DataSource`.)
3. Explicitly initialize the `WebCore` specifying *`"http"`* or *`"https"`* as `AssetProtocol`.
4. Create a `WebSession` and add your custom `DataSource` using `CATCH_ALL` as host name.
5. Create a new `WebView` using the created `WebSession`.

   Now, all requests for and from remote pages (using the *`"http"`* or *`"https"`* protocol), irrespective of hostname (domain name), will be directed to your custom *catch-all* `DataSource`. But you cannot (and you don't want to) provide all resources from local assets.

6. Implement `IResourceInterceptor` and assign your implementation to `WebCore.ResourceInterceptor`. Even when you use DataSources, all requests targeting a `DataSource` are still passing from `IResourceInterceptor` before reaching `DataSource.OnRequest`.
7. In your implementation of `IResourceInterceptor.OnRequest`, check the value of `ResourceRequest.Url` (or any other parameters you want to evaluate). If the request targets a resource that you want to load asynchronously, set `IgnoreDataSources` to *`false`*; for all other requests, set `IgnoreDataSources` to *`true`*. `IgnoreDataSources` tells Awesomium to ignore any DataSources registered for this asset protocol and hostname, and process the request normally (which means the request will be sent to the remote server).

This way you can load only certain resources asynchronously while let the rest of the resources be normally loaded from the remote server.

For a sample of this scenario, see the Windows Forms *WinFormsSample* available with the SDK. The sample code (see: *`WebForm.cs`*) can be used with any technology and web-view component (WPF, MonoMac etc.).

New Features

Core

- Code optimizations and performance improvements all throughout the **Awesomium.NET** project.
- Added logging of **Awesomium.NET** events to application log.
- Improved initial position *specs* for JavaScript `window.open` calls.
- Improved JIF to assist native `DocumentReady`, improve `HTML` property contents and handle pending `window.close` calls.
- Added **PDF.js** integration.
- Added support for environment variables in `DataPath`, `LogPath` etc.
- Improved performance of `WebCore.QueueWork`.
- Added support for JavaScript `navigator.language`.
- Added support for DOM `HTMLAnchorElement.click`.
- Added standard *`onXXXX`* touch event handler setting properties to JavaScript `Node` prototype.
- Improved `JSValue` **->** `bool` operator to process all types.
- Added binary and unary operators to `JSValue`.
- `JSValue` is now a class.
- Many performance improvements in `ResourceInterceptor`.
- Added support for asynchronous `ResourceInterceptor` responses through `ResourceRequest.IgnoreDataSources` and DataSources.
- Completely redesigned (re-wrote) JIF using supported ECMAScript 5 features.
- Designed and created the `OSMJIF` instance that exposes a fully operational Javascript Framework to JavaScript clients.
- `OSMJIF` API allows clients obtain global and per-view information.
- `OSMJIF` API allows clients add listeners for global or per-view native events.
- `OSMJIF` API allows clients control parts of the native application.
- `OSMJIF` extends the DOM to handle all multi-touch-related features (such as scrolling).
- `OSMInfo`, `OSMEventArgs` and `OSMView` fully configured JavaScript prototypes part of the new API.
- Made it so Javascript-related events and custom JavaScript method handlers are called in a **Javascript Execution Context (JEC)**.
- `JSObject` instances acquired or created in a **Javascript Execution Context (JEC)** don't need to explicitly disposed.
- Errors or exceptions that occur in a **Javascript Execution Context (JEC)** are silently propagated to the JavaScript console.
- Made it so most `JSObject` operations are first handled by JIF, if available, significantly improving performance.
- Made it so implicit casting of `JSValue` to `JSObject` or `JSFunction` always succeeds returning an invalid object.
- Added full support for dynamically indexing JSObjects.
- Made it so local JSObjects hold members in internal managed dictionaries.
- Made it so all dynamic expressions on `JSObject` return either `JSValue` or `JSObject`.
- Many optimizations and performance improvements on `JSObject`.

`JSObject` is not sealed any more.
- Added `JSObject` indexer overloads that take a `JSPropertyDescriptor`.
- `JSObject` is now enumerable (enumerates ECMAScript enumerable property names).
- Made it so users can restore local JSObjects back to their original subclass when reacquired from V8.
- Most late binding errors on `JSObject` (in DLR) no longer throw an exception.
- Added dynamic conversion support to `JSObject`.
- Added support for passing managed handlers as callbacks directly to dynamic expressions.
- Added basic support of DLR to `JSValue`.
- Added indexer to `JSValue` that allows accessing and editing arrays, objects and strings.
- Added support for passing objects through the synchronous `OSMJIF.sendMessage`.
- Made it so all methods or events executed in a JEC have access to essential JavaScript objects (through `Global`).
- More **VB.NET** DLR improvements.
- A single background thread is now handling auto-update in `InAutoUpdate` mode.
- Significantly improved and added more documentation.

WPF

- Added *`http://schemas.awesomium.com/core`* **XMLNS** schema for Core assembly.
- Added WPF Designer extensions for `WebControl` and `WebSessionProvider` for Visual Studio 2010, 2012 and 2013.
- Added full WPF Touch/Stylus support.
- Made it so we cancel touch manipulation when Javascript is disabled.
- Made it so mouse is only injected on tap or after press-and-hold.
- Added support for scrolling (touch-drag) parent scrollable elements instead of the view.

Unity

- Added `WebSessionProvider` component.
- Implemented Undo/Redo for Inspector changes.

API Changes

New API:

Awesomium.Core

- `DocumentReadyState`
- `DocumentReadyEventArgs`
- `DocumentReadyEventHandler`
- `WebTouchEvent`
- `WebTouchEventType`
- `WebTouchPoint`
- `WebTouchPointState`
- `NativeHandle`
- `WebCore.DoWork`

- `WebCore.UsedMemory`
- `WebCore.StartTime`
- `WebCore.ReleaseMemory`
- `WebCore.AllocatedMemory`
- `WebCore.UsedMemory`
- `WebCore.Run(CoreStartEventHandler)`
- `WebConfig.ASSET_PROTOCOL_DEFAULT`
- `WebConfig.CustomCSS`
- `WebPreferences.MaxHttpCacheStorage`
- `WebPreferences.PdfJS`
- `WebPreferences.UserScript`
- `WebPreferences.JavascriptViews`
- `WebPreferences.JavascriptApplicationInfo`
- `WebPreferences.JavascriptGlobalEvents`
- `WebPreferences.JavascriptViewEvents`
- `WebPreferences.JavascriptViewExecute`
- `WebPreferences.JavascriptViewChangeSource`
- `JavascriptRequest`
- `JavascriptMessageEventArgs`
- `JavascriptRequestEventArgs`
- `JavascriptMessageEventHandler`
- `JavascriptRequestEventHandler`
- `JavascriptExecutionContextMethod`
- `JavascriptExecutionContextMethod<T>`
- `IWebView.JavascriptRequest`
- `IWebView.JavascriptMessage`
- `IWebView.CreateJavascriptExecutionContext`
- `IWebView.InjectTouchEvent`
- `IWebView.CreationTime`
- `JSFunction`
- `JSObject.GetPropertyDescriptor`
- `JSObject[..., JSPropertyDescriptor]`
- `JSObject.BindAsync`
- `JSValue[...]`
- `JSValue.IsNaN`
- `JSValue.IsInfinity`
- `JSValue.IsFunctionObject`
- `JSValue (All Unary & Binary operators)`
- `Global`
- `DocumentReadyEventArgs.Environment`
- `JavascriptMethodHandler`
- `JavascriptAsyncMethodHandler`
- `JSFunctionHandler`
- `JSFunctionAsyncHandler`

- `JSPropertyDescriptor`
- `ResourceRequest.IgnoreDataSources`
- `WebContextMenuInfo.IsEmpty`

Awesomium.Windows.Controls (WPF)

- `WebControlService.PressAndHoldDelay`
- `Utilities.SetCulture`

Awesomium.Windows.Forms (Windows Forms)

- `Utilities.SetCulture`

JavaScript

- `OSMJIF`
- `OSMInfo`
- `OSMEventArgs`
- `OSMView`

Modified API:

Awesomium.Core

- `IWebView.DocumentReady`
- `IWebView.Instance` -> `NativeHandle`
- `JSValue` ***struct*** -> ***class***

Obsolete API:

Awesomium.Core

- `JSObject.Bind(String,Boolean,JavascriptMethodEventHandler)`
- `JavascriptMethodEventHandler`
- `JavascriptAsynchMethodEventHandler`

Bug Fixes

Native Awesomium

- Fixed crash on Windows XP when using Facebook Connect (and other sites with similar certificate signing modes).
- Fixed crash that occurs if user unfocuses a textbox during an IME composition.
- Fixed crash with very large strings of `WebConfig.UserScript`.

Awesomium.Core

- Fixed issue in `ResourceDataSource` (folders with a dash are replaced by an underscore).
- Assigning null string to `JSValue` doesn't set `JSValue.IsNull`. (#19)
- Fixed invalid synchronization context issues that may occur on *UpdateTimerCallback*, when `UpdateState` is `InAutoUpdate`. (#48)
- Fixed illegal disposal of the managed wrapper of the global `Null` and `Undefined` JSValues. (#53)
- Fixed issues with `IWebView.HTML` not returning the full page contents. (#61)
- Fixed issue where early *`window.close`* calls where not being processed. (#61)
- Using invoke on several threads crashes Awesomium. (#59)
- Fixed issues with `JSObject` dynamic indexer calls on **VB.NET**.
- Fixed issues with `IWebView.SaveImageAt`.
- Fixed issue with `JSObject.ToString` crashing when called on Global objects.
- Fixed issue in `WebSession.HasViews` that could cause a `WebSession` being prematurely released.
- Fixed issue where local `JSObject` instances being assigned as members of other local JSObjects, are set by value.
- Fixed massive thread spawning at `InAutoUpdate` mode when UI thread is blocked. (#71)
- Fixed issue where `SurfaceFactory` would fail to destroy unused surfaces.
- Fixed issue preventing navigation when only the anchor of a URL is changed. (#52)

Awesomium.Windows.Controls (WPF)

- Fixed `ArgumentException` at `RenderProcess` getter. (#69)
- Fixed exception that occurred during the initialization of `WebControlCommands`. (#57)
- Fixed error in `ISynchronizeInvoke.InvokeRequired` implementation.
- Fixed issue with `DataPakSourceProvider.PakPath` validation.
- Fixed issue that would prevent temporarily unloaded `WebControl` containers (such as those in a TabControl's tab), accessing the `WebDialogsLayer` decorator.

Changes in Samples

- Updated all samples to use a standard `WebCore.Shutdown` policy.
- Fixes and improvements in ***BasicAsyncSample*** to use latest features.
- Updated ***JavascriptSample*** to reflect the new **JEC** features and new **DLR** support features.
- Updated ***WinFormsSample*** demonstrating asynchronously loading resources through ResourceInterceptor+DataSource.
- Updated ***TabbedWPFSample*** demonstrating how to extend **JIF**.
- Updated WPF ***WebControlSample*** to reflect the new **JEC** features and new **DLR** support features.
- Updated ***WPFJavascriptSample*** to reflect the new **JEC** features and new **DLR** support features.
- Expanded WPF ***StarterSample*** and ***VBStarterSample*** with examples of taking screenshot and interacting with the page.
- Updated WPF ***StarterSample*** and ***VBStarterSample*** to reflect the new **JEC** features and new **DLR** support features.

Known Issues

- On **Windows 8**, **WebGL** is currently not supported.
- **Multi-touch** support for WPF `WebControl` on **Windows 7** and earlier, lacks the *press-and-hold* feature. This is a system limitation. However, events are still propagated to the mouse if you keep your finger to the surface until the system's *press-and-hold* gesture completes.
- Manipulating **scrollbars** when you are using a **touch** surface or stylus with the WPF `WebControl`, is currently handled as regular touch manipulation and may have the oposite from the expected effect. To manipulate scrollbars with a touch point as you would with a mouse, use the *press-and-hold* gesture. (You can still however scroll pages using normal touch gestures, inside the page.)

Under production:

- Design-time support of the WPF `WebSessionProvider` currently does not allow editing the `DataSources` list directly through the respective dialog but this feature is planned for next release.
- When you are using the OSX `OSMWebView`, drop-down (popup) menus (e.g., HTML: `<select>`), are not displayed automatically. Predefined drop-down (popup) menus have been added to the WPF `WebControl` and the Windows Forms `WebControl` but not to the MonoMac `OSMWebView` yet. However, the new powerful API allows you to design and display these yourself, by handling the `ShowPopupMenu` event.

Older Changelogs

You can find this and previous Changelogs, under the Changelogs category.