WebKitGTK+ Programming Guide

Version 0.1

Bob Murphy

1 December 2011

Table of Contents

Introduction....2
- Prerequisites....2
- Installation....2
  - Pre-Built....2
  - From Source Code....2

Tutorial....3
- One-Window Browser....3

Cookbook....5
- Monitoring Load Status....5
- Interacting with the DOM: Finding Text Elements....5
- Language Bindings....6
  - Vala....6
  - Mono....7
  - Python....7

Reference....8
- Class Hierarchy....8
- Working with the DOM....8
- Selected APIs....9
  - Visual Appearance....9
  - Events....9
  - Finding Elements in the DOM....11
  - Working with Node Lists....13
  - Miscellaneous....13
- Other References....13

Introduction

WebKitGTK+ is the version of the WebKit open-source web engine that uses GTK+ as its user-facing front-end. It's a powerful system with a rich array of functionality – but like many such systems, without explanation, it can have a very steep learning curve.

This document is intended to help you learn to use WebKitGTK+ in a basic way, provide some code recipes and examples, and especially, to give you a general orientation to the code base and other documentation.

Prerequisites

This document assumes you have a good working knowledge of:

- The C programming language
- The GObject system, including types, signals, and properties
- GTK+
- The *WebKitGTK+ Reference Manual* (http://webkitgtk.org/reference/webkitgtk/stable/index.html). You don't need to understand everything, but you should at least skim the page about WebKitWebView, which is the central class of the WebKitGTK+ API.

WebKit itself is written in C++, so if you plan to work with the system's internals, knowledge of that language is also important.

If you plan to write code that interacts with the content of web pages, you should also be familiar with the HTML *Document Object Model*, or *DOM***.**

Installation

Pre-Built

WebKitGTK+ is available as an installable developer package for most major Linux distributions. This is the best option if you are just beginning to work with WebKit, or if you are not experienced with build systems that have complex dependencies.

From Source Code

If you prefer to build WebKitGTK+ yourself, you will need to:

1. Obtain the sources: http://webkitgtk.org/?page=download
2. Build them: http://webkitgtk.org/?page=contribute

Depending on how you obtain your sources, when you do the build you may need to use `configure` in place of `autogen.sh`. You may also need to build or install other development packages that WebKitGTK+ depends on; the WebKitGTK+ build will notify you if any required packages are missing.

Building WebKitGTK+ can take a long time, especially since some parts of it cannot be built in parallel.

Tutorial

One-Window Browser

The following is a very simple example of how to use WebKitGTK+. It simply creates a single window containing a browser instance that displays a single web page.

```
/*
 * Copyright (C) 2006, 2007 Apple Inc.
 * Copyright (C) 2007 Alp Toker <alp@atoker.com>
 * Copyright (C) 2011 Lukasz Slachciak
 * Copyright (C) 2011 Bob Murphy
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY APPLE COMPUTER, INC. ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL APPLE COMPUTER, INC. OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
 * OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <gtk/gtk.h>
#include <webkit/webkit.h>


static void destroyWindowCb(GtkWidget* widget, GtkWidget* window);
static gboolean closeWebViewCb(WebKitWebView* webView, GtkWidget* window);

int main(int argc, char* argv[])
{
    // Initialize GTK+
    gtk_init(&argc, &argv);

    // Create an 800x600 window that will contain the browser instance
    GtkWidget *main_window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_default_size(GTK_WINDOW(main_window), 800, 600);

    // Create a browser instance
    WebKitWebView *webView = WEBKIT_WEB_VIEW(webkit_web_view_new());

    // Create a scrollable area, and put the browser instance into it
    GtkWidget *scrolledWindow = gtk_scrolled_window_new(NULL, NULL);
    gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(scrolledWindow),
            GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);
    gtk_container_add(GTK_CONTAINER(scrolledWindow), GTK_WIDGET(webView));

    // Set up callbacks so that if either the main window or the browser instance is
    // closed, the program will exit
    g_signal_connect(main_window, "destroy", G_CALLBACK(destroyWindowCb), NULL);
    g_signal_connect(webView, "close-web-view", G_CALLBACK(closeWebViewCb), main_window);

    // Put the scrollable area into the main window
    gtk_container_add(GTK_CONTAINER(main_window), scrolledWindow);
```

```
    // Load a web page into the browser instance
    webkit_web_view_load_uri(webView, "http://www.webkitgtk.org/");

    // Make sure that when the browser area becomes visible, it will get mouse
    // and keyboard events
    gtk_widget_grab_focus(GTK_WIDGET(webView));

    // Make sure the main window and all its contents are visible
    gtk_widget_show_all(main_window);

    // Run the main GTK+ event loop
    gtk_main();

    return 0;
}


static void destroyWindowCb(GtkWidget* widget, GtkWidget* window)
{
    gtk_main_quit();
}

static gboolean closeWebViewCb(WebKitWebView* webView, GtkWidget* window)
{
    gtk_widget_destroy(window);
    return TRUE;
}
```

Once you're comfortable that you understand this example, you may want to review the more complicated GtkLauncher example program. This is contained in the WebKit source code, in the Tools directory, and demonstrates more features of WebKitGTK+, such as how work with the browser history.

Cookbook

Monitoring Load Status

```
// Attach a load status listener to a web view
g_signal_connect(webView, "notify::load-status", G_CALLBACK(loadStatusCb), context);

...

// Implement the listener, handling each possible load status
static void loadStatusCb(WebKitWebView *web_view, GParamSpec *pspec, void* context)
{
    WebKitLoadStatus status = webkit_web_view_get_load_status (web_view);
    GObject *object = G_OBJECT (web_view);
    g_assert(web_view == context->m_WebView);

    g_object_freeze_notify (object);

    switch (status)
    {
    case WEBKIT_LOAD_FINISHED:
        handleLoadFinished(web_view, context);
        break;

    case WEBKIT_LOAD_PROVISIONAL:
        handleLoadProvisional(web_view, context);
        break;

    case WEBKIT_LOAD_COMMITTED:
        handleLoadCommitted(web_view, context);
        break;

    case WEBKIT_LOAD_FIRST_VISUALLY_NON_EMPTY_LAYOUT:
        handleLoadNonEmptyLayout(web_view, context);
        break;

    case WEBKIT_LOAD_FAILED:
        handleLoadFailed(web_view, context);
        break;

    default:
        break;
    }

    g_object_thaw_notify (object);
}
```

Interacting with the DOM: Finding Text Elements

This is an example that works with the *Monitoring Load Status* example above. When a web page has finished loading, it looks through the DOM for the loaded page to find all text and password nodes, and attaches listeners for focus and blur events on those nodes.

```
static void myFocusCallback (WebKitDOMNode *element, WebKitDOMEvent *dom_event,
                  void* context)
{
    ...
}

static void myBlurCallback (WebKitDOMNode *element, WebKitDOMEvent *dom_event,
                  void* context)
```

```
{
    ...
}

static void handleLoadFinished(WebKitWebView *web_view, void* context)
{
    WebKitDOMDocument *document =
            webkit_web_view_get_dom_document(WEBKIT_WEB_VIEW(web_view));
    WebKitDOMNodeList *elements =
            webkit_dom_document_get_elements_by_tag_name(document, "*");
    gulong element_count = webkit_dom_node_list_get_length(elements);

    for (int i = 0; i < element_count; i++)
    {
        WebKitDOMNode *element = webkit_dom_node_list_item(elements, i);

        if (WEBKIT_DOM_IS_HTML_INPUT_ELEMENT (element))
        {
            char *element_type;
            g_object_get (element, "type", &element_type, NULL);

            if (g_str_equal ("text", element_type) ||
                    g_str_equal("password", element_type))
            {
                webkit_dom_event_target_add_event_listener(
                        WEBKIT_DOM_EVENT_TARGET (element), "focus",
                        G_CALLBACK (myFocusCallback), false,
                        context);
                webkit_dom_event_target_add_event_listener(
                        WEBKIT_DOM_EVENT_TARGET (element), "blur",
                        G_CALLBACK (prvBlurCallback), false,
                        context);

            }
            g_free (element_type);
        }
    }

    g_object_unref(elements);


    // Is there already an active element? Sometimes this happens, and no
    // focus event is emitted afterward. This handles that situation.
    WebKitDOMHTMLDocument *htmlDocument = WEBKIT_DOM_HTML_DOCUMENT(document);
    WebKitDOMElement* activeElement =
            webkit_dom_html_document_get_active_element(htmlDocument);
    if (WEBKIT_DOM_IS_HTML_INPUT_ELEMENT (activeElement))
    {
        char *activeElementType = NULL;
        g_object_get (activeElement, "type", &activeElementType, NULL);
        if (g_str_equal ("text", activeElementType) ||
                g_str_equal("password", activeElementType))
        {
            myFocusCallback(WEBKIT_DOM_NODE(activeElement), NULL, context);
        }
        g_free(activeElementType);
    }
}
```

Language Bindings

WebKitGTK+ has bindings for several languages other than C.

Vala

You can see a sample Vala program which uses WebKitGTK+ at http://live.gnome.org/Vala/WebKitSample

Mono

Mono bindings for WebKitGTK+ are provided by webkit-sharp: http://www.mono-project.com/Libraries#WebKit_Sharp

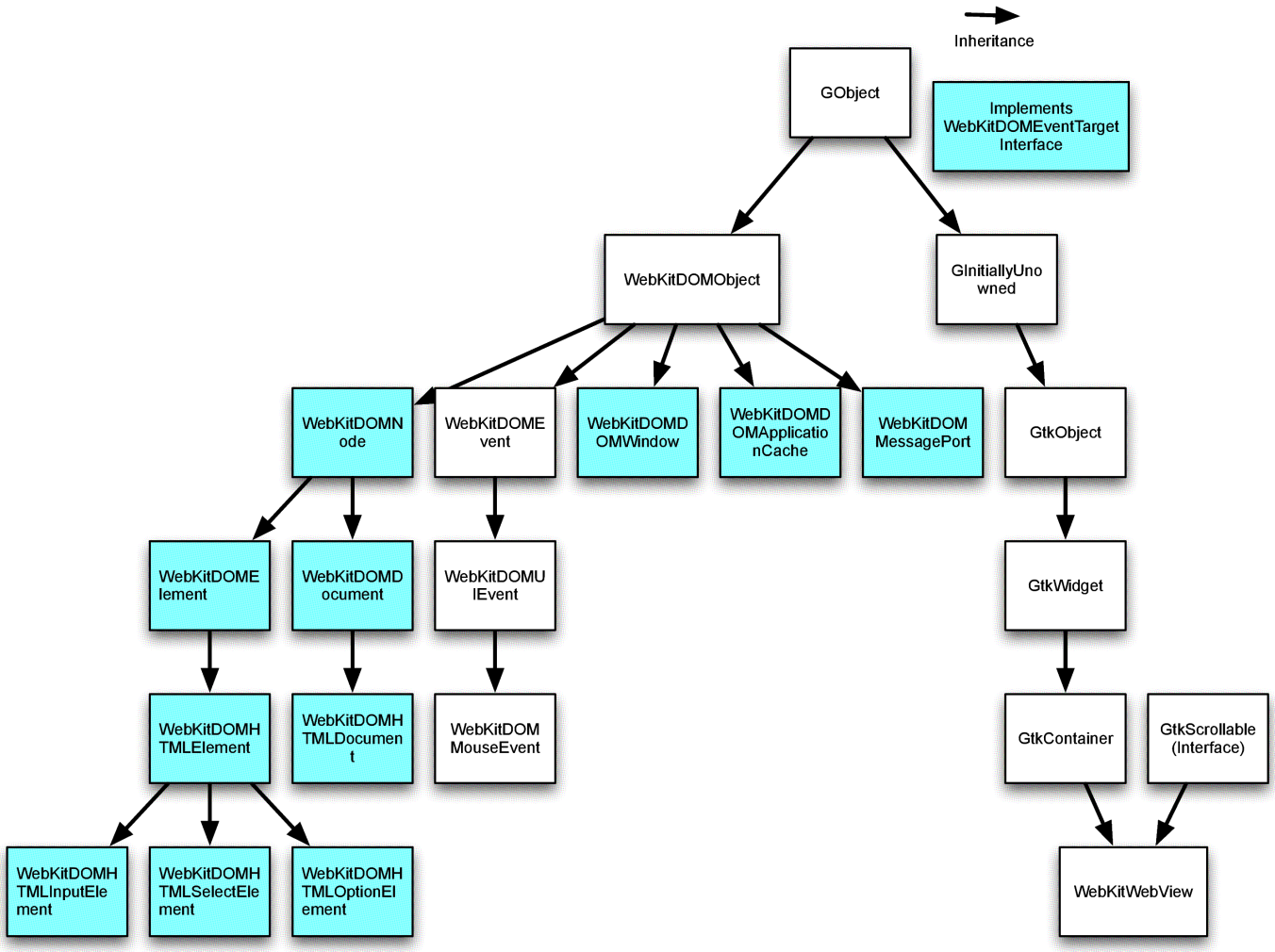
Python

Python bindings for WebKitGTK+ are hosted at http://code.google.com/p/pywebkitgtk/

Reference

Class Hierarchy

The diagram below shows the GObject inheritance hierarchy for some of the more important WebKitGTK+ object classes.



Working with the DOM

The *WebKitGTK+ Reference Manual* covers the GTK+ specific interface for WebKit. However, most of the functionality provided by WebKitGTK+ is related to working with the DOM, which is not documented in that manual. Fortunately, there's a workaround for that.

The DOM specification is defined by the W3C in the form of Interface Definition Language (IDL) files, which are included as part of the WebKit source code. The WebKit build system uses these to create the C-language DOM API bindings for WebKitGTK+, as well as the JavaScript bindings, and other graphical front-ends for WebKit such as Objective C bindings for Mac OS X and iOS.

While the GTK+ DOM binding isn't well documented, the IDL itself and the JavaScript binding are very thoroughly documented – see:

- The W3C's IDL specifications: http://www.w3.org/TR/#tr_DOM
- Mozilla's JavaScript DOM reference: https://developer.mozilla.org/en/Gecko_DOM_Reference
- w3schools' JavaScript references and tutorials: http://www.w3schools.com/htmldom/dom_intro.asp

That being the case, it's often easiest to look at the IDL or JavaScript documentation, find a reference to the DOM functionality you need, and then figure out the equivalent WebKitGTK+ API.

For instance, let's say you want to know if the user is holding the control key down when doing a mouse click. The references above show a `ctrlKey` attribute in the IDL for `MouseEvent`, and in JavaScript, if you have an event named `event`, you can test `event.ctrlKey`. If you go to where your WebKitGTK+ headers are installed, and search for "ctrl", you will find that WebKitDOMMouseEvent.h contains:

```
WEBKIT_API gboolean
webkit_dom_mouse_event_get_ctrl_key(WebKitDOMMouseEvent* self);
```

which is exactly the API call you need. In general, if you are looking for a DOM method `fooBar`, you can easily find the equivalent WebKitGTK+ C function by searching the header files for "foo" or "bar", and the resulting function prototype will contain `foo_bar`.

Selected APIs

WebKitGTK+ supports over 1800 different function calls; this document only covers a limited subset.

Visual Appearance

webkit_dom_element_scroll_into_view()

```
void webkit_dom_element_scroll_into_view (WebKitDOMElement* self,
                gboolean align_with_top);
```

This will scroll the specified element into view. If the `align_with_top` argument is true, the scrolled element will be aligned with the top of the scroll area; otherwise, it will be aligned with the bottom.

webkit_dom_element_scroll_into_view_if_needed()

```
void webkit_dom_element_scroll_into_view_if_needed(WebKitDOMElement* self, gboolean
center_if_needed);
```

This will scroll an element into view if it is outside the viewport. If the `center_if_needed` argument is true, this will center the element in the viewport; otherwise, it will scroll as little as possible to make the element visible in the viewport.

Unlike `webkit_dom_element_scroll_into_view()` , this will not scroll any elements if the argument element is already fully visible in the viewport.

Events

There are many interesting prototypes in WebKitDOMEvent.h; here are a few.

webkit_dom_event_get_type()

```
GType webkit_dom_event_get_type (void);
```

Returns the GObject type of a DOM event.

webkit_dom_event_stop_immediate_propagation()

```
void webkit_dom_event_stop_immediate_propagation(WebKitDOMEvent* self);
```

If you call this from inside an event listener, it will stop further propagation of the event in the current phase.

webkit_dom_event_target_dispatch_event()

```
void webkit_dom_event_target_dispatch_event(WebKitDOMEventTarget *target,
                                            WebKitDOMEvent       *event,
                                            GError              **error);
```

GObjectEventListenerCallback

```
typedef void (*GObjectEventListenerCallback)  (GObject*,
                                               WebKitDOMEvent*,
                                               void*);
```

This is the actual callback type used by `webkit_dom_event_target_add_event_listener()` and `webkit_dom_event_target_remove_event_listener()` below.

webkit_dom_event_target_add_event_listener()

```
gboolean webkit_dom_event_target_add_event_listener(WebKitDOMEventTarget *target,
                                                    const char           *eventName,
                                                    GCallback             handler,
                                                    boolean               bubble,
                                                    gpointer              userData);
```

`bubble` indicates whether the handler should be called during the *capture* or *bubble* phase of event propagation. You can register the same function for both phases, and it's treated as two different handlers for purposes of both being called, and for removal - see webkit_dom_event_target_remove_event_listener() below.

Returns: TRUE if the listener was successfully added.

webkit_dom_event_target_remove_event_listener()

```
gboolean webkit_dom_event_target_remove_event_listener(WebKitDOMEventTarget *target,
                                                       const char           *eventName,
                                                       GCallback             handler,
                                                       gboolean              bubble);
```

Removes a previously-added event listener.

Returns: TRUE if the listener was successfully removed.

webkit_dom_event_get_target()

```
WebKitDOMEventTarget* webkit_dom_event_get_target(WebKitDOMEvent* self);
```

Returns: the element the event actually occurred on. This does not vary with event capturing and bubbling.

webkit_dom_event_get_current_target()

```
WebKitDOMEventTarget* webkit_dom_event_get_current_target(WebKitDOMEvent* self);
```

Returns: the element whose event listeners are currently being processed. This changes during event capturing and bubbling.

webkit_dom_event_get_event_phase()

```
gushort webkit_dom_event_get_event_phase(WebKitDOMEvent* self);
```

Indicates which phase of the event flow is currently being evaluated.

Returns: 1 if in capturing phase
2 if at the target
3 if in bubbling phase

Finding Elements in the DOM

webkit_dom_document_get_elements_by_tag_name()

```
WebKitDOMNodeList* webkit_dom_document_get_elements_by_tag_name(WebKitDOMDocument* self,
                                                                const gchar* tagname);
```

Searches the subtree beneath (but not including) *self*, and returns a list of elements with the given tag name. The special value "*" matches all tags.

```
WebKitDOMNodeList* list = webkit_dom_document_get_elements_by_tag_name(doc, “td”);
```

webkit_dom_document_get_elements_by_tag_name_ns()

```
WebKitDOMNodeList* webkit_dom_document_get_elements_by_tag_name_ns(
                                    WebKitDOMDocument* self,
                                    const gchar* namespace_uri,
                                    const gchar* local_name);
```

Searches the subtree beneath (but not including) *self*, and returns a list of elements with the given namespace and tag name. The special value "*" matches all namespaces and tags, respectively.

```
WebKitDOMNodeList* list = webkit_dom_document_get_elements_by_tag_name(doc,
                                "http://www.w3.org/1999/xhtml", “td”);
```

webkit_dom_document_get_elements_by_name()

```
WebKitDOMNodeList* webkit_dom_document_get_elements_by_name(WebKitDOMDocument* self,
                                                            const gchar* element_name);
```

Searches the subtree beneath (but not including) *self*, and returns a list of elements with the given name attribute.

```
WebKitDOMNodeList* up_elems = webkit_dom_document_get_elements_by_name(doc, “up”);
```

webkit_dom_document_get_element_by_id()

```
WebKitDOMElement* webkit_dom_document_get_element_by_id(WebKitDOMDocument* self,
                                                        const gchar* element_id);
```

Returns the element with the specified ID, or NULL if no such element exists. If more than one element has this ID, the behavior is undefined.

webkit_dom_document_get_elements_by_class_name()

```
WebKitDOMNodeList* webkit_dom_document_get_elements_by_class_name(
                                    WebKitDOMDocument* self,
                                    const gchar* tagname);
```

Searches the subtree beneath (but not including) *self*, and returns a list of elements with the given class name. Multiple class names can be included by using spaces as separators. For instance:

```
WebKitDOMNodeList* elems = webkit_dom_document_get_elements_by_class_name(doc,
            “red blue”);
```

will retrieve all elements that have both red and blue classes.

webkit_dom_document_get_forms()

```
WebKitDOMHTMLCollection* webkit_dom_document_get_forms(WebKitDOMDocument* self);
```

Returns a list of the form elements within *self*.

webkit_dom_element_get_elements_by_tag_name()

```
WebKitDOMNodeList* webkit_dom_element_get_elements_by_tag_name(WebKitDOMElement* self,
                                                               const gchar* name);
```

Searches the subtree beneath (but not including) *self*, and returns a list of elements with the given tag name. The special value "*" matches all tags.

webkit_dom_element_get_elements_by_tag_name_ns()

```
WebKitDOMNodeList* webkit_dom_element_get_elements_by_tag_name_ns(
                                  WebKitDOMElement* self,
                                  const gchar* namespace_uri,
                                  const gchar* local_name);
```

Searches the subtree beneath (but not including) *self*, and returns a list of elements with the given namespace and tag name. The special value "*" matches all namespaces and tags, respectively.

webkit_dom_element_get_elements_by_class_name()

```
WebKitDOMNodeList* webkit_dom_element_get_elements_by_class_name(
                                  WebKitDOMElement* self,
                                  const gchar* name);
```

Searches the subtree beneath (but not including) *self*, and returns a list of elements with the given class name. Multiple class names can be included by using spaces as separators.

webkit_dom_html_document_has_focus()

```
gboolean webkit_dom_html_document_has_focus(WebKitDOMHTMLDocument* self);
```

Returns whether or not the document itself, or an element in the document, has focus.

webkit_dom_html_document_get_active_element()

```
WebKitDOMElement* webkit_dom_html_document_get_active_element(
                                  WebKitDOMHTMLDocument* self);
```

Returns the active element in the document - e.g., either the focused element, or the element that would receive focus if the document itself had focus. For an element to be focused, it must be active, and its document must have focus.

Note: When a document first loads, sometimes an element receives focus without a focus event firing.

webkit_dom_html_form_element_get_elements()

```
WebKitDOMHTMLCollection* webkit_dom_html_form_element_get_elements(
                                    WebKitDOMHTMLFormElement* self);
```

Returns a collection of all the form control elements in the form.

Working with Node Lists

webkit_dom_node_list_get_length()

```
gulong webkit_dom_node_list_get_length(WebKitDOMNodeList* self);
```

Get the number of items in the list.

webkit_dom_node_list_item()

```
WebKitDOMNode webkit_dom_node_list_item(WebKitDOMNodeList* self, gulong index);
```

Get the list's node at the specified index.

Miscellaneous

webkit_dom_html_document_get_active_element()

```
WebKitDOMElement* webkit_dom_html_document_get_active_element(
                                    WebKitDOMHTMLDocument* self);
```

Get the active element in the specified document.

You can obtain a document to be used here by calling `webkit_web_view_get_dom_document()` on a WebKitWebView, and then dynamically casting the return value to a `WebKitDOMHTMLDocument`.

Other References

- An Apple paper on WebKit DOM programming from a JavaScript perspective is available at http://developer.apple.com/library/mac/documentation/AppleApplications/Conceptual/SafariJSProgTopics/SafariJSProgTopics.pdf
- Another Apple paper on using WebKit from Objective C (e.g. for Mac OS X or iOS) is at http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/DisplayWebContent/WebKit_DisplayWebContent.pdf. This paper contains numerous how-tos and recipes which demonstrate functionality that can also be used from WebKitGTK+.
- The *Hacker's guide to WebKit/GTK+* provides an overview of the WebKit source tree from a GTK perspective, along with other interesting information: http://trac.webkit.org/wiki/HackingGtk
- Information on how to contribute to WebKit itself is available at http://www.webkit.org/coding/contributing.html, and links some articles on technical topics related to WebKit internals are at http://www.webkit.org/coding/technical-articles.html