TkGecko: Another Attempt for an HTML Renderer for Tk

Georgios Petasis

Software and Knowledge Engineering Laboratory,
Institute of Informatics and Telecommunications,
National Centre for Scientific Research “Demokritos”,
Athens, Greece
petasis@iit.demokritos.gr

Abstract

The support for displaying HTML and especially complex Web sites has always been problematic in Tk. Several efforts have been made in order to alleviate this problem, and this paper presents another (and still incomplete) one. This paper presents TkGecko, a Tcl/Tk extension written in C++, which allows Gecko (the HTML processing and rendering engine developed by the Mozilla Foundation) to be embedded as a widget in Tk. The current status of the TkGecko extension is alpha quality, while the code is publically available under the BSD license.

1 Introduction

The support for displaying HTML and contemporary Web sites has always been a problem in the Tk widget, as Tk does not contain any support for rendering HTML pages. This shortcoming has been the motivation for a large number of attempts to provide support from simple rendering of HTML subsets on the text or canvas widgets (i.e. for implementing help systems) to full-featured Web browsers, like HV3 [1] or BrowseX [2]. The relevant Tcl Wiki page [3] lists more than 20 projects, and it does not even cover all of the approaches that try to embed existing browsers in Tk through COM or X11 embedding.

One of the most popular, and thus important, projects is Tkhtml [4], an implementation of an HTML rendering component in C for the Tk toolkit. Tkhtml has been actively maintained for several years, and the current version supports many HTML 4 features, including CCS and possibly JavaScript through the Simple ECMAScript Engine (SEE) [5]. Despite the impressive supported list of features, Tkhtml is still missing important features found in contemporary Web sites, such as complete JavaScript, Flash or Java support. Thus, several approaches that try to embed a full-featured Web browser and instrument it through various means have been presented. For the Windows operating system it is quite easy to embed and instrument Internet Explorer using COM, either through Tcom [6] or Optcl [7]. Under Unix, a similar effort has been made by the Entice project [8; 9], which embeds Firefox through the TkXext [10] extension for X11. The extension presented in this paper, TkGecko, is more similar to the latter set of approaches. It does not implement another HTML renderer, but tries to reuse an existing one. Aiming at supporting a wide range of operating systems and, at the same time, ensure high degrees of compatibility, the rendering engine of the cross-platform and popular Web browser Firefox was chosen, known as Gecko [11].

It is not the first time that the embedding of Gecko in Tcl/Tk has been attempted. An earlier project under the same name (TkGecko) existed nearly a decade ago, company-sponsored, whose results and progress remain unknown as it was a closed-source project. The main sources of information regarding this earlier attempt are a paper presented at the 7th USENIX Tcl/Tk conference [12] and references in the Tcl Wiki [13]. However there is nothing common of the TkGecko presented in this paper and the earlier approach except the name: the two projects do not share any code, while the newer TkGecko is an open source project, distributed under the BSD license and whose source code is publically available at SourceForge [14].

The rest of the paper is organised as follows: chapter two provides an overview of the TkGecko extension, along with an example of how the extension can be initialised and used in order to render a Web page. Chapter three briefly presents the widget subcommands, along with the facilities for accessing the DOM tree of the rendered page. Finally, chapter four concludes this document and presents some future directions.

2 TkGecko: Embedding Gecko in Tk

Mozilla's Gecko rendering engine is not a simple piece of code. Being the essential element of many products (including Firefox, Thunderbird, Camino, Flock, SeaMonkey, k-Meleon, Netscape 9 and many others), Gecko is a cross-platform, standards-compliant and feature-complete rendering engine. Its complexity is evident not only by the size of its source code, but also by the effort required in order to embed it in a C++ application. Embedding and communication with the Gecko engine is performed through a protocol known as XPCOM [16], a cross-platform component object model, similar to Microsoft's COM. There is documentation on how Gecko can be embedded in C++ applications [16], but there are several problems when such an embedding is attempted:

- Stability of the API: the public API is more fluid than stable, and currently there are several embedding APIs available. Despite that TkGecko being less than a year old, a novel embedding API has been presented already [17], making the embedding process a moving target.
- Complexity of the API: several classes have to be implemented, as functionality is scattered among many interfaces that usually interconnect with each other.
- Thread-safety: the threading model followed by the Gecko engine internally is unknown, making difficult to ensure thread safety if the host application also uses threads.
- Dependence upon toolkits: under some platforms (especially Linux), the Gecko engine relies on widget toolkits (such as GTK) for rendering content and widgets. This poses many additional problems, as the toolkit must be also initialised by the host application, its message loop initialised, and native windows must be mapped as windows of the toolkit.

TkGecko implements a large percent of the classes required, offering access to a wide range of features from the Tcl/Tk programming language. Its source code is a mixture of static code and code dynamically generated through SWIG [18] in order to handle the number of the interfaces that must be exposed as Tcl commands. Despite its complexity, though, building TkGecko from sources is not so difficult (provided that a Development XULRunner SDK is available) as it is handled by the CMake build tool. Regarding the aforementioned problems the stability and complexity of the API have been coded with the hope that the used API will not be deprecated for a while. Dependence upon the GTK toolkit is something that affects only Linux, and the current implementation initialises GTK and initiates its event loop in a separate thread automatically by the TkGecko extension, while Gecko is loaded and initialised in the thread that also runs GTK. Finally, thread-safety is still a target to be reached: despite the code has been developed with thread-safety in mind, stability in a single-threaded host application is still an issue after prolonged use.

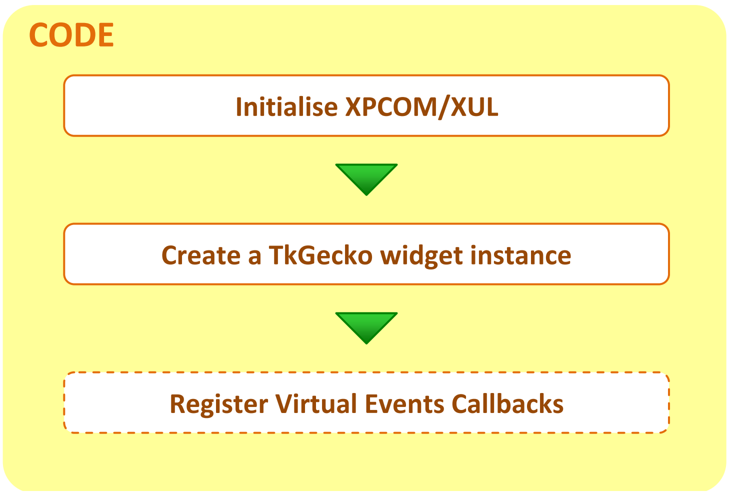

CODE
Initialise XPCOM/XUL
Create a TkGecko widget instance
Register Virtual Events Callbacks

Figure 1: Steps in creating a TkGecko widget.

2.1 Using TkGecko: an example

In order to use the TkGecko extension, the steps shown in Figure 2 must be followed. The first step needs to be done only once during the lifetime of an application, and simply involves pointing the TkGecko extension the location of the XPCOM shared library and initialising the XUL layer. The XPCOM shared library can be usually found inside a XULRunner distribution, or inside the directories of products that use Gecko, such as Mozilla Firefox. Once the location (or possible locations) of the XPCOM shared library have been identified, the command "tkgecko::initializeXPCOM *path1 … pathN*" can be used to initialise the XPCOM layer, as shown in Figure 2. The command "tkgecko::initializeXPCOM" will return the full path name of the XPCOM shared library actually loaded, in case this is interesting to the caller.

```
package require Tk
package require tkgecko

set paths {
  {C:\Program Files (x86)\Mozilla Firefox\xpcom.dll}
  {C:\Program Files\Mozilla Firefox\xpcom.dll}
  /usr/lib64/xulrunner-1.9.1/libxpcom.so
  /usr/lib64/xulrunner-sdk-1.9.1/sdk/lib/libxpcom.so
  /usr/lib/xulrunner-1.9.1/libxpcom.so
  /usr/lib/xulrunner-sdk-1.9.1/sdk/lib/libxpcom.so
}
set xpcom [tkgecko::initializeXPCOM {*}$paths]
puts "XPCOM library: $xpcom"
```

Figure 2: Initialising the XPCOM layer.

After XPCOM has been initialised, the XUL layer must be also initialised. XUL stands for "XML User Interface Language" [20], and its description is beyond the scope of this paper. The interested user is referred to the "Mozilla Developer Center" [21]. An easy work-around is to use the XUL offered by the XULRunner or the product containing the XPCOM shared library:

```
set xuldir [file nativename [file dirname $xpcom]]
set appdir {} ;# Same as xuldir...
set profiledir [file native [file normalize ~/.tkgecko]]
puts "XUL directory:      $xuldir"
puts "APP directory:      $appdir"
puts "Profile directory: $profiledir"
puts "tkgecko::initializeXUL:\
  [tkgecko::initializeXUL $xuldir $appdir $profiledir]"
puts =========================================================
puts "              Initialisation completed!"
puts =========================================================
```

Figure 3: Initialising the XUL layer.

2.2 TkGecko: creating a widget

Having initialised the XPCOM and XUL layers (step 1), we are ready to create a widget. Let's create a minimal browser then (Figure 4):

```
set URI https://developer.mozilla.org/en-US/
grid [ttk::button .back    -text { < }     -command onBack] \
     [ttk::button .forward -text { > }     -command onForward] \
     [ttk::button .reload  -text {Reload}  -command onReload] \
     [ttk::entry  .uri     -textvariable URI] \
     [ttk::button .load    -text {Load}    -command onLoad] \
  -padx 2 -pady 2 -sticky snew
grid [tkgecko::browser .browser -width 800 \
                -height 600 -highlightthickness 1] \
  -columnspan 5 -sticky snew -padx 2 -pady 2
grid [ttk::label .status -textvariable STATUS] - - -\
     [ttk::progressbar .progress] \
  -sticky snew -padx 2 -pady 2

grid columnconfigure . 3 -weight 1
grid rowconfigure    . 1 -weight 1
```

Figure 4: A minimal Web Browser – a TkGecko widget and a few buttons.

2.3 TkGecko: adding bindings

Communication between a TkGecko widget and the host application is performed through Tk virtual events. The TkGecko extension defines many virtual events, which are delivered to the TkGecko widget when an event has occurred in the Gecko instance (i.e. the loading of a Web page has been completed). When a virtual event is delivered, additional data related to the event will be appended to it, and can be retrieved with the "%d" code in the binding script. Since the list of supported virtual events is lengthy and the event names are self-explanatory, we will briefly present them here through some Tcl code, continuing our example (Figure 5):

```
##
## Bindings:
##
bind .browser <<OnStatusScriptChange>> {set ::STATUS [lindex %d 0]}
bind .browser <<OnStatusLinkChange>>   {set ::STATUS [lindex %d 0]}
bind .browser <<OnStatusChange>>       {set ::STATUS [lindex %d 0]}
bind .browser <<OnSetTitle>>           {wm  title   . [lindex %d 0]}
bind .browser <<OnProgressChange>>     {onProgress {*}%d}

## Other virtual events...
# bind .browser <<OnLocationChange>>     {}
# bind .browser <<OnSetDimensions>>      {}
# bind .browser <<OnStop>>               {}
# bind .browser <<OnStateChange>>        {}
# bind .browser <<OnFocusNextElement>>   {}
# bind .browser <<OnFocusPrevElement>>   {}
# bind .browser <<OnSetFocus>>           {}
# bind .browser <<OnRemoveFocus>>        {}
# bind .browser <<OnVisibilityChange>>   {}
# bind .browser <<OnShowTooltip>>        {}
# bind .browser <<OnHideTooltip>>        {}

bind .browser <<OnDocumentLoadInit>>     {onLoadInit   {*}%d}
bind .browser <<OnDocumentLoadFinish>>   {onLoadFinish {*}%d}
```

Figure 5: An example of bindings that can be bound on a TkGecko widget.

The code that implements the callback functions is shown in Figure 6.

```
proc onLoadInit {args} {
  puts "<<onLoadInit>>: $args"
  .progress state !disabled
  .progress configure -maximum 100 -value 0
};# onLoadInit

proc onLoadFinish {args} {
  puts "<<onLoadFinish>>: $args"
  .progress state disabled
  update idle
  after 1000 {set ::STATUS {}}
  testDOM
};# onLoadFinish

proc onBack {} {
  .browser back
};# onBack

proc onForward {} {
  .browser forward
};# onForward

proc onReload {} {
  .browser reload
  onNewPage
};# onReload

proc onLoad {} {
  .browser navigate $::URI
  onNewPage
};# onLoad

proc onNewPage {} {
  if {[.browser can_go_back]} {
    .back state !disabled} else {.back state disabled}
  if {[.browser can_go_forward]} {
    .forward state !disabled} else {.forward state disabled}
};# onNewPage

proc onProgress {uri curUriProgress   maxUriProgress
                   curTotalProgress maxTotalProgress} {
  # puts "$curTotalProgress $maxTotalProgress"
  set curTotalProgress [expr {abs($curTotalProgress)}]
  set maxTotalProgress [expr {abs($maxTotalProgress)}]
  if {$maxTotalProgress >= $curTotalProgress} {
    .progress configure -maximum $maxTotalProgress \
                        -value $curTotalProgress
  }
};# onProgress
```

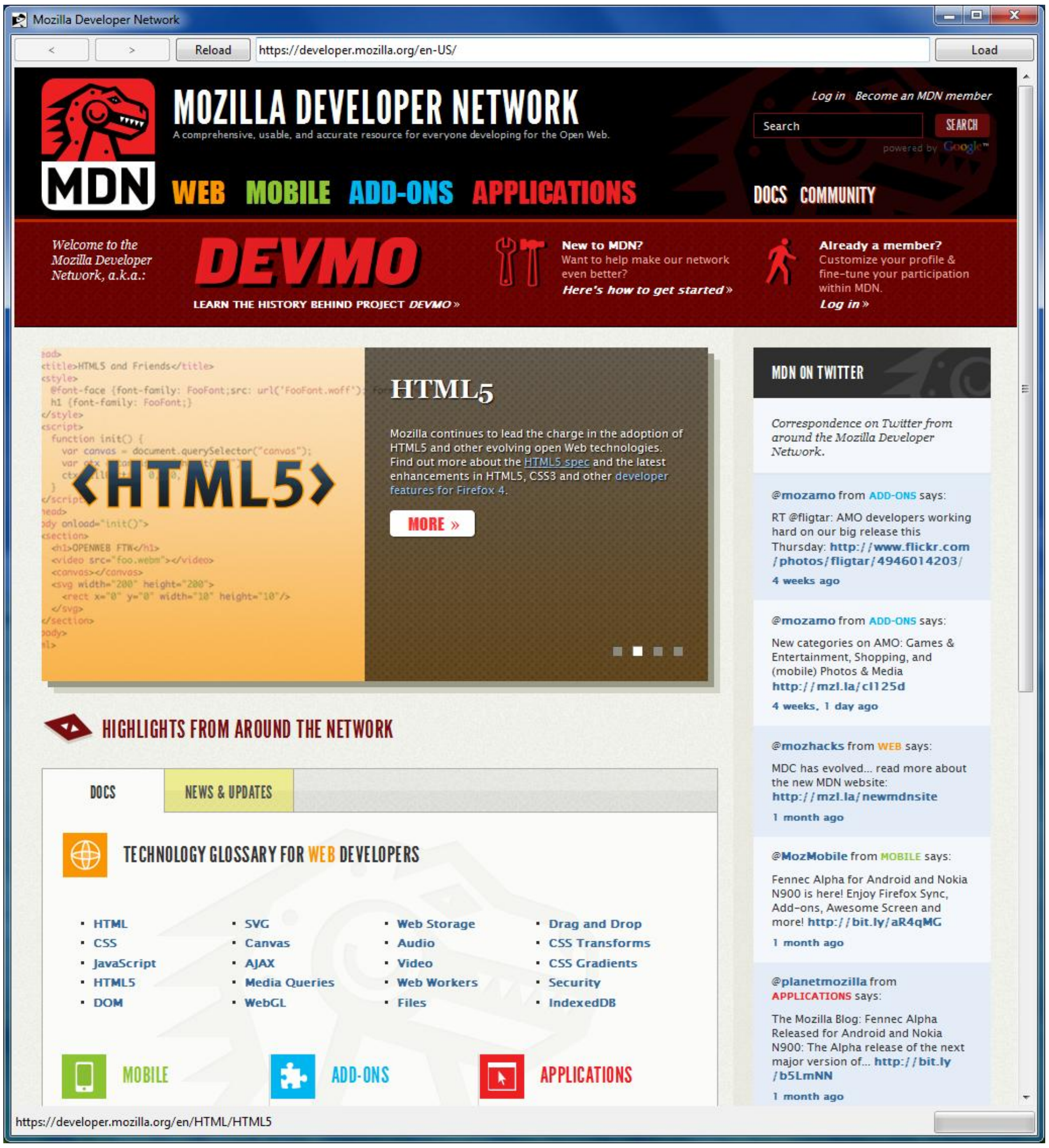
Figure 6: The binding callbacks for the minimal browser example.

Finally, some initialisation code finalises the example (Figure 7):

```
bind all <Key-Return> onLoad
onLoad
focus .uri
```

Figure 7: Loading the default URI in the TkGecko widget.

Running the example produces the following output (Figure 8):



Mozilla Developer Network
<
>
Reload
https://developer.mozilla.org/en-US/
Load
MOZILLA DEVELOPER NETWORK
A comprehensive, usable, and accurate resource for everyone developing for the Open Web.
Log in Become an MDN member
Search
SEARCH
powered by Google
MDN
WEB MOBILE ADD-ONS APPLICATIONS
DOCS COMMUNITY
Welcome to the Mozilla Developer Network, a.k.a.:
DEVMO
LEARN THE HISTORY BEHIND PROJECT DEVMO »
New to MDN?
Here's how to get started »
Already a member?
Log in »
HTML5
MORE »
HIGHLIGHTS FROM AROUND THE NETWORK
DOCS
NEWS & UPDATES
TECHNOLOGY GLOSSARY FOR WEB DEVELOPERS
MOBILE
ADD-ONS
APPLICATIONS
MDN ON TWITTER
https://developer.mozilla.org/en/HTML/HTML5

Want to help make our network even better?

Customize your profile & fine-tune your participation within MDN.

Mozilla continues to lead the charge in the adoption of HTML5 and other evolving open Web technologies. Find out more about the HTML5 spec and the latest enhancements in HTML5, CSS3 and other developer features for Firefox 4.

- HTML
- CSS
- JavaScript
- HTML5
- DOM
- SVG
- Canvas
- AJAX
- Media Queries
- WebGL
- Web Storage
- Audio
- Video
- Web Workers
- Files
- Drag and Drop
- CSS Transforms
- CSS Gradients
- Security
- IndexedDB

Correspondence on Twitter from around the Mozilla Developer Network.

@mozamo from ADD-ONS says:
RT @fligtar: AMO developers working hard on our big release this Thursday: http://www.flickr.com/photos/fligtar/4946014203/
4 weeks ago

@mozamo from ADD-ONS says:
New categories on AMO: Games & Entertainment, Shopping, and (mobile) Photos & Media http://mzl.la/cl125d
4 weeks, 1 day ago

@mozhacks from WEB says:
MDC has evolved... read more about the new MDN website: http://mzl.la/newmdnsite
1 month ago

@MozMobile from MOBILE says:
Fennec Alpha for Android and Nokia N900 is here! Enjoy Firefox Sync, Add-ons, Awesome Screen and more! http://bit.ly/aR4qMG
1 month ago

@planetmozilla from APPLICATIONS says:
The Mozilla Blog: Fennec Alpha Released for Android and Nokia N900: The Alpha release of the next major version of... http://bit.ly/b5LmNN
1 month ago

Figure 8: The minimal Web browser in action.

3 TkGecko widget subcommands

Events that occur inside the TkGecko widget area, owned by the Gecko rendering engine, are communicated to the host application through Tk virtual events. On the other hand, the host application can deliver events to the Gecko engine by invoking subcommands of the TkGecko widget instance. Currently, the following subcommands are supported:

- *pathname* **back**: this subcommand instructs the widget to load the previously displayed Web page. Requires history to be enabled.
- *pathname* **can_go_back**: this subcommand returns true if history is enabled, and at least one Web page is contained in the history. In all other cases false is returned.
- *pathname* **can_go_forward**: this subcommand returns true if history is enabled, and at least one Web page is contained in the history after the currently displayed page. In all other cases false is returned.
- *pathname* **document**: this subcommand returns the name of a Tcl command, that represents the DOM object of the current page.
- *pathname* **expose**: this subcommand delivers an expose event to the Gecko rendering engine.
- *pathname* **focus_activate**: this subcommand transfers focus from the host application to the Gecko engine.
- *pathname* **focus_deactivate**: this subcommand transfers focus from Gecko engine to the host application.
- *pathname* **forward**: this subcommand instructs the widget to load the next Web page in the history list, if such page exists. Requires history to be enabled.
- *pathname* **hide**: this subcommand hides the page rendering from the widget.
- *pathname* **load**: this subcommand loads the Web page specified by the current URI.
- *pathname* **linkmsg**: this subcommand returns the URI of the link below the mouse cursor.
- *pathname* **navigate URI ?flags?**: this subcommand navigates the widget to the specified URI. Flags can be an OR value from the values of the following variables: ::tkgecko::nsIWebNavigation_LOAD_FLAGS_BYPASS_CACHE,
  ::tkgecko::nsIWebNavigation_LOAD_FLAGS_BYPASS_HISTORY,
  ::tkgecko::nsIWebNavigation_LOAD_FLAGS_BYPASS_PROXY,
  ::tkgecko::nsIWebNavigation_LOAD_FLAGS_CHARSET_CHANGE,
  ::tkgecko::nsIWebNavigation_LOAD_FLAGS_IS_LINK,
  ::tkgecko::nsIWebNavigation_LOAD_FLAGS_IS_REFRESH,
  ::tkgecko::nsIWebNavigation_LOAD_FLAGS_MASK,
  ::tkgecko::nsIWebNavigation_LOAD_FLAGS_NONE,
  ::tkgecko::nsIWebNavigation_LOAD_FLAGS_REPLACE_HISTORY.
- *pathname* **parse ?-base base_uri? ?-mime mime_type? ?--? data**: this subcommand displays the string provided through the **data** parameter.
- *pathname* **reload**: this subcommand reloads the current Web page.
- *pathname* **save ?-data_dir data_dir? ?-mime mime_type? ?-flags flags? ?-pflags persist_flags? ?-col wrap_col? ?--? uri**: this subcommand saves the current Web page.
- *pathname* **setup nsIWebBrowserSetup::Id boolean**: this subcommand changes a property of the Gecko rendering engine.
- *pathname* **show**: this subcommand shows the page rendering into the widget.
- *pathname* **special**: this subcommand is reserved for internal use of the TkGecko extension.
- *pathname* **statusmsg**: this subcommand returns the text that should be displayed in the status bar of the widget. The text returned may be the result of executing JavaScript code.
- *pathname* **stop ?flags?**: this subcommand stops the operation described by the specified flags. Flags can be an OR value from the values of the following variables: ::tkgecko::nsIWebNavigation_STOP_ALL,
  ::tkgecko::nsIWebNavigation_STOP_CONTENT,

::tkgecko::nsIWebNavigation_STOP_NETWORK.
If no flags are provided, the default value is ::tkgecko::nsIWebNavigation_STOP_ALL.

- *pathname* **title**: this subcommand returns the text that should be displayed in the title bar of the toplevel containing the widget. The text returned may be the result of executing JavaScript code.
- *pathname* **uri ?uri?**: this subcommand returns the current URI, if no additional arguments are specified. If an **uri** is provided, it sets the current URI to the specified one.
- *pathname* **visibility**: this subcommand returns true if the rendering is visible in the widget window, false otherwise.

3.1 TkGecko and the DOM tree

TkGecko provides extensive support for the DOM tree kept by the Gecko engine for the current page, exposing it at the Tcl level as Tcl commands. This part of the TkGecko is automatically generated through SWIG [18], thus it mimicking the exact API offered by the Gecko XPCOM objects. Currently, the classes supported are:

- nsIDOMHTMLCollection
- nsIDOMNodeList
- nsIDOMNamedNodeMap
- nsIDOMNode
- nsIDOMElement
- nsIDOMHTMLElement
- nsIDOMAttr
- nsIDOMDocument
- nsIDOMHTMLDocument
- nsIWebBrowserPersist
- nsIDocumentEncoder
- nsIWebBrowserSetup
- nsIWebNavigation

Interested readers may consult the Mozilla Developer Center and the Embedding API in order to get information about the usage of these classes and their methods. In this subsection we are going to present only some basic examples, just to illustrate the usage of the provided API from Tcl. The first example involves retrieving the HTML code of the currently rendered page, which may be different than its HTML code, due to the execution of JavaScript. Assuming that the variable “browser” contains the pathname of a TkGecko widget, the HTML of the currently rendered page can be retrieved with the following code (Figure 9):

```
set dom     [$browser document]
set body    [$dom GetBody]
set content [$dom SerializeToString $body]
$body -delete
$dom  -delete
```

Figure 9: Retrieving the HTML code of the current page. The HTML code will be saved in the “content” variable.

Another simple example is to convert the currently rendered page in plain text, using an encoder. The example is shown in Figure 10.

```
set dom     [$browser document]
set body    [$dom GetBody]
set encoder [$dom GetEncoder text/plain 0]
if {$encoder ne "NULL"} {
  $encoder SetNode $body
  set content [$encoder EncodeToString]
  $encoder SetNode NULL
} else {
  set content "NULL encoder!"
}
$body    -delete
$encoder -delete
$dom     -delete
```

Figure 10: Retrieving the contents of the current page as plain text. The result will be saved in the "content" variable.

4 Conclusions

This paper presents TkGecko, an extension written in C++ for the Tk toolkit that embeds Mozilla's rendering engine, Gecko, as a Tk widget. The extension allows the rendering of pages with the same compliance as the popular Firefox Web browser, supporting features such as CSS, HTML 5, Flash, JavaScript or Java. TkGecko is an open source extension distributed under the BSD license and hosted at the SourceForge repositories [14]. The current status of TkGecko is of alpha quality, as some stability issues observed during prolonged use (such as memory leaks and random locks after displaying a few tenths of Web pages) need to be resolved. Future work on TkGecko will concentrate on identifying and fixing all these issues, so as to achieve the first stable release of TkGecko. Currently TkGecko has been tested only on the Microsoft Windows and GNU/Linux operating systems. It is planned to additionally perform tests under the Apple OS X operating system. Finally, the suitability of the new embedding API described at [17] for the purposed of TkGecko will be evaluated.

5 References

[1] Hv3 - Tcl/Tk Web Browser: http://tkhtml.tcl.tk/hv3.html
[2] BrowseX: http://pdqi.com/browsex/
[3] Tk Web Browsers: http://wiki.tcl.tk/3134
[4] Tkhtml: a Tk widget that displays content formatted according to the HTML and CSS standards. http://tkhtml.tcl.tk/index.html
[5] SEE: Simple ECMAScript Engine: http://www.adaptive-enterprises.com.au/~d/software/see/
[6] TkXext: http://wiki.tcl.tk/TkXext
[7] OpTcl: http://www2.cmp.uea.ac.uk/~fuzz/optcl/default.html
[8] Landers S., 2006: "Entice: Multiple Firefox instances in a Tk frame". In Proceedings of the 13th Annual Tcl/Tk Conference (Tcl'2006), Naperville, Illinois, USA, 9–13 October, 2006.
[9] Landers S., 2007: "Entice – Embedding Firefox in Tk". In Proceedings of the 14th Annual Tcl/Tk Conference (Tcl'2007), New Orleans, Louisiana, USA, 24–28 September, 2007.
[10] TkXext: http://wiki.tcl.tk/TkXext
[11] Mozilla Gecko: http://en.wikipedia.org/wiki/Gecko_(layout_engine)
[12] Ball S., 2000: "TKGECKO: A Frill-Necked Lizard". In Proceedings of the 7th USENIX Tcl/Tk Conference (Tcl'00), Austin, Texas, USA, 14–18 February, 2000.
[13] Tcl Wiki: http://wiki.tcl.tk
[14] TkGecko SourceForge Project page: https://sourceforge.net/projects/tkgecko/
[15] XPCOM: https://developer.mozilla.org/en/XPCOM
[16] Embedding Mozilla; https://developer.mozilla.org/en/Embedding_Mozilla
[17] Mozilla's Gecko Embedding/NewApi: https://wiki.mozilla.org/Embedding/NewApi
[18] SWIG – Simplified Wrapper and Interface Generator: http://www.swig.org/
[19] CMake – Cross Platform Make: http://www.cmake.org/
[20] XUL – XML User Interface Language: https://developer.mozilla.org/En/XUL, http://en.wikipedia.org/wiki/XUL
[21] Mozilla Developer Network: https://developer.mozilla.org/en-US/