JavaScript/QML extension support in Falkon

Name: Anmol Gautam
Email: tarptaeya@gmail.com
IRC Nick: tarptaeya
Mentor: David Rosca
Location: Gandhinagar, India (GMT+0530)

Introduction

Falkon is a web browser, previously known as Qupzilla. Falkon supports extensions but in C++ & Python. The task for the project is to integrate QJSEngine and QQmlEngine with Falkon, so that extensions can be developed using JavaScript with QML for GUI.

Current state of extension support in Falkon

Falkon supports two type of extensions:

- Compiled C++ extensions which are being loaded from predefined libraries.
- PySide2/Python scripting to interact with the browser API.

Since Falkon supports compiled C++ extensions, it's both difficult (comparatively) to develop and distribute. Other mainstream browsers use JavaScript with XML (or XML like) language for extension (plugin) support which makes it easy to develop and distribute as they can be loaded dynamically into the browser.

A typical Falkon plugin has the following structure:

- The plugin spec which contains general information about plugin (author, version, description, etc)
- Interaction with Plugin API (actions, load-unload, etc.)

Also the loaded plugin is displayed under the tools button in the navigation bar.

My proposal

Aim: To provide API which allows to develop extensions using *JavaScript* and *QML*, so that user can download them and load them dynamically into Falkon - which makes extensions both easy to develop and distribute.

- The QML plugins will be loaded from standard plugin paths (from subdirectory qml) which is similar as the python plugins.
- The information about all the plugins is to be stored in *metadata.desktop* file of each plugin , which contains the spec for the downloaded plugin and permissions(to load in incognito, etc).
- I will try to make the API for the following browser interactions:
    - actions & events - to interact with main browser actions and events like *mousePress*, *mouseMove*, *wheelEvent*, *keyPress*,etc.
    - browser state - to interact with the windowState of browser like *isFullScreen*, *triggerFullScreen*, *isMinimised*, etc.
    - notifications - to show desktop notifications.
    - permissions - to interact with browser permissions.
    - scheme handling - which handles a specific URL scheme.
    - settings - to interact with browser settings.
    - sidebar - to add new sidebar into application.
    - signals - to interact with browser signals emitted on creation/deletion of WebPage and MainWindow objects.
    - tabs - to interact with Tab functions (or WebTab's) like *getTabCount*, *unloadTab*, *muteTab*, etc.
    - web browsing patterns (top sites, bookmarks, history, downloads, etc).
- GUI for the plugin, if required by the plugin, can be in form of a toolbar button (just like adblock button) or address bar button (for the case of developing a reading mode plugin ...) which will show GUI and context menu on requested.
- I will also make "falkon:extensions" as a URL scheme to display the information about plugins (by information I mean settings, current state, license etc) and the option to enable/disable the plugin rather than using *Menu -> Preferences -> Extensions*.

Currently themes are different from plugins in Falkon. A typical Falkon theme has the following structure:

- main.css - contains main stylesheet for Falkon.
- theme-name.desktop - contains theme info.
- rtl.css - (optional) stylesheet for RTL languages.
- license.txt - (optional) license file of theme, with no default license.

- theme.png - a 32x32 icon for theme.

I would also add theme management to "falkon:extensions" as a different navigation tab ("falkon:extension" will have two navigation tabs one for web-extensions and the other for theme - similar to what firefox does using "about:addons"). Also the theme license defaults to GPLv3 if not provided by the author.

Implementation details and design

- As with C++ and Python extensions, the entry point of QML plugins will be PluginInterface class. The QQmlEngine will be used to load and instantiate the plugin object in the function *Plugins::loadAvailablePlugins()*.
- The main QML plugin will have the similar structure as C++/Python extension.
- The API to interact with the main browser functions can be implemented mainly as wrapper objects.
- Translations will be done by exporting i18n/i18np C++ functions to QML engine that handles the translations with gettext module of python.
- If the plugin requires GUI, it will be implemented in QML and integration into the QtWidgets can be done using QQuickWidget.

The typical plugin will look somewhat like (As suggested by my mentor, David Rosca) -

```
import org.kde.falkon as Falkon

Falkon.PluginInterface {
  init: function(state, settingsPath) {
    print("init", state, settingsPath);
  }
  unload: function() { }
  // ... etc
}
```

Tentative timeline

Dates	Schedule
till 14th May	I will explore more about integrating C++/Qt/JS on how to make Applications scriptable using js and knowing kde infrastructure and scripting.
14th May - 20th May	developing basic structure of QML plugin in Falkon. At this stage qml files will be loaded into Falkon and can be registered as plugins. The plugin at this stage will be without interaction API, just load/unload, init, specs, ...
20th May	From this I will start developing the interaction API for Falkon.
20th May - 30th May	will add events, actions to API
30th May - 5th June	will add browser state and notification to API
6th June - 10th June	will add scheme handling and tabs to API
11th June - 15th June	will add settings to API
16th June - 25th June	will implement signals and permissions. I will also begin thinking for GUI. I will implement sidebar API after implementing GUI for the plugin.
26th June - 10th July	will implement GUI for the both the plugin in the toolbar and for the sidebar
11th July - 15th July	will implement sidebar API
15th July - 20th July	will implement the API functions which might come to my mind while developing.
21st July - 25th July	will implement the "falkon:extensions" scheme and will complete to parse theme as plugins. Themes management will also be added to "falkon:extensions". At this stage almost all work is complete.
25th July - 31st July	will fix bugs that might come while development.

1st August - 10th August	finalize documentation based on notes developed alongside the code

Besides this tentative timeline, I will write unit-tests and documentation notes alongside writing the code and will integrate all the notes into a complete documentation for writing QML plugins for Falkon.

I will try to communicate daily with my mentor. Also I don't have any commitments for the summer. My end semester exams will be finished by 30th April and the new term will start from the last week of July so I can easily devote about 40 hours/week working on the project.

About me

I am a 19 years old 2nd year undergraduate with majors in computer science at Indian Institute of Technology, Gandhinagar. I have a good experience with Qt/C++, JavaScript, OCaml, Python and PHP. Previous summer I made a web browser in Qt5 - Crusta http://crustabrowser.com. I have also developed some open source JavaScript libraries (idetext.js) and some are under development (lona.js, context.js). I am a big lover of Open Source. Besides KDE I also contribute to OCaml. Link to my Github profile is https://github.com/tarptaeya.
I also qualified for ACM ICPC regionals in 2016 for Kolkata site and in 2017 for Kharagpur and Chennai sites, beside winning other local algorithmic competitions.
My contributions to KDE are predominantly in Falkon. The links to the merged patches to Qupzilla (now Falkon) are -

- resolved issue #2608 - Printing to PDF with the page title #2609
- fixed drop url over unrestored tabs #2604
- removed unnecessary cursor:pointer from the body of speeddial #2599
- fixed tab ToolTip on pinning/unpinning #2598
- Bookmarks ToolBar refresh fix #2594

The links to merge patches on Falkon via Phabricator -

- Clear mouseover state after closing menu in bookmarks toolbar
- Fix losing menubar visibility settings after quit from fullscreen [BUG: 391415]
- SpeedDial: Add 'overflow-y: auto' for '#overlay-edit'
- VerticalTabs: Open new tab with double and middle click in free space

No, I am not applying for any other organisation or project for GSoC.