

1. User Guide	2
2. API overview	4
2.1 addon - xml Definition	4
2.1.1 addon.background	5
2.1.1.1 addon.background.script	5
2.1.2 addon.browser	5
2.1.2.1 addon.browser.chrome	5
2.1.2.2 addon.browser.firefox	6
2.1.2.3 addon.browser.ie	6
2.1.2.4 addon.browser.opera	6
2.1.2.5 addon.browser.safari	7
2.1.3 addon.content	7
2.1.3.1 addon.content.script	7
2.1.4 addon.interface	7
2.1.5 addon.key	8
2.1.6 addon.stat	8
2.1.7 addon.ui	8
2.1.7.1 addon.ui.button	9
2.1.7.2 addon.ui.icons	9
2.1.7.2.1 addon.ui.icon	9
2.1.7.3 addon.ui.settings	10
2.1.8 addon.update	10
2.1.8.1 addon.update.chrome	11
2.1.8.2 addon.update.firefox	11
2.1.8.3 addon.update.ie	11
2.1.8.4 addon.update.opera	11
2.1.8.5 addon.update.safari	12
2.2 framework - JavaScript Interface	12
2.2.1 framework.browser	12
2.2.1.1 framework.browser.attachEvent()	12
2.2.1.2 framework.browser.navigate()	13
2.2.2 framework.extension	13
2.2.2.1 framework.extension.attachEvent()	13
2.2.2.2 framework.extension.author	14
2.2.2.3 framework.extension.description	14
2.2.2.4 framework.extension.detachEvent()	14
2.2.2.5 framework.extension.fireEvent()	15
2.2.2.6 framework.extension.getId()	15
2.2.2.7 framework.extension.getItem()	15
2.2.2.8 framework.extension.getRequest()	16
2.2.2.9 framework.extension.log()	16
2.2.2.10 framework.extension.name	16
2.2.2.11 framework.extension.setItem()	17
2.2.2.12 framework.extension.stat()	17
2.2.2.13 framework.extension.updateUrl	17
2.2.2.14 framework.extension.url	17
2.2.2.15 framework.extension.version	18
2.2.3 framework.ui	18
2.2.3.1 framework.ui.button	18
2.2.3.1.1 framework.ui.button.attachEvent()	18
2.2.3.1.2 framework.ui.button.setBadgeBackgroundColor()	18
2.2.3.1.3 framework.ui.button.setBadgeColor()	19
2.2.3.1.4 framework.ui.button.setBadgeText()	19
2.2.3.1.5 framework.ui.button.setIcon()	19
2.2.3.1.6 framework.ui.button.setPopup()	20
2.2.3.1.7 framework.ui.button.setTitle()	20
2.2.3.2 framework.ui.toolbar	20
3. Known Issues	20

User Guide

Add-ons Framework

The instruction on how to build the extensions for Chrome, Opera, Safari, Internet Explorer and Firefox browsers

(C) 2011

Contents

- Description
- File structure
- Getting started
- Building via command line
- Structure of a configuration file
- Extensions rebuilding (update)
 - Chrome
 - Internet Explorer
- Tuning OS Windows to run Windows Host Script
 - OS Windows Vista/7
 - OS Windows XP
- Difference between "background" and "content" scripts
- License Information

Description

Add-ons Framework is intended to create extensions for main Internet browsers (Chrome, Opera, Safari, Internet Explorer and Firefox). A user writes scripts on JavaScript using Add-on Framework API. Add-on Framework builds a ready to use extension for certain browsers.

File structure

pack\utils – system tools and scripts

pack\custom – user created js-scripts, html-pages, pictures, style sheets, etc.

pack\chrome – an output folder with Chrome add-on's files (the folder is used only by the builder)

pack\opera – an output folder with Opera add-on's files (the folder is used only by the builder)

pack\IE – an output folder with Internet Explorer add-on's files (the folder is used only by the builder)

pack\Safari – an output folder with Safari add-on's files (the folder is used only by the builder)

pack\Firefox – an output folder with Firefox add-on's files (the folder is used only by the builder)

At the moment following rules must be observed:

1. All user created files should be placed in a folder called "custom";
2. It is necessary to consider that all used js-files will be processed by an obfuscator. In a Debug-mode user's scripts remain without any changes;
3. Content of the "custom" folder will be copied to add-on «as is», except for js-files and configuration files.

Getting Started

The sequence of actions to create an add-on using the framework is following:

1. Create "background" and "content" scripts using API (see "API specifications" section). Save scripts to files, or import them into a configuration file in CDATA section. To understand a difference between "background" and "content" scripts see "Difference between *background* and *content* scripts" section;
2. Create a page for popup, if it is necessary;
3. Create icons in PNG format or import them to a configuration file in base64 format. For Internet Explorer extension icon must be ICO format;
4. Adjust a configuration file "config.xml" (see "Configuration file structure" section)
5. Start the builder using "pack.bat"
6. If on start an error occurs, for more details see "OS Windows setup to run Windows Host" section;
7. On success, add-on files will be created (for Chrome it is a *.crx file, for Opera - *.oex, for Internet Explorer - *.exe, for Safari - *.safariextz, for Firefox - *.xpi). File name is taken from a "config.xml" file.

Building via command line

As the builder represents a JavaScript-file, script running works under control of Windows Host Script. To build via the command line do the following:

1. Run the Command line;
2. Go to the directory 'pack';
3. Type "**cscript utils\pack.js**" and then press Enter.

Command line flags:

- **-DEBUG** – The builder doesn't use a code obfuscation for user scripts to detect any possible errors while developing an extension.

The configuration file structure

Configuration file is set with XML format. More details you can find in [Api Overview](#) section.

- config.xml

This file contains full description of created extension. You can set name, version, content of extension etc.

Packer builds an extension based on content of this file.

Extensions rebuilding (update)

For Chrome and Internet Explorer extensions, before rebuilding it is required to save keys generated on initiating building of an extension.

Chrome

After building an extension Chrome creates a *.pem key. This key is saved in a "custom" folder with a name, same as extension name (it is taken from "config.xml", "addon" node, "filename" attribute)

When rebuilding an extension, if the given file is found it is used for update.

Internet Explorer

Binary files of the given extension contain GUID. For various extensions these keys differ. Therefore, when building a new extension, new GUIDs are generated.

New GUIDs are saved in a *.guid file with a file name same as in an extension (it is taken from "config.xml", "addon" node, "filename" attribute)

When rebuilding an extension, if the given file is found, guid located in this file is used for update.

Tuning OS Windows to run Windows Host Script

OS Windows Vista/7

1. Run command: "**regsvr32 %systemroot%\system32\jscript.dll**";
2. Run script **vista_js_fix.reg** (utils\reg\vista_js_fix.reg).

OS Windows XP

1. Run command: "**regsvr32 %systemroot%\system32\jscript.dll**";
2. Run script **xp_js_fix.reg** (utils\reg\xp_js_fix.reg).

Difference between "background" and "content" scripts

Content scripts are the ones that run in a content of web pages. By using the Document Object Model (DOM), they can read or make changes in web pages. The background script is a script that runs in the extension process. It exists for the lifetime of your extension. Only one instance of it at a time can be active.

License Information

Copyright(C)2011 Besttoolbars

Add-on Framework is distributed under the Non-commercial Software License Agreement.

License permits free of charge use on non-commercial and private web sites only under special conditions (as described in the license). There are also Commercial Software Licenses available.

Please visit http://besttoolbars.net/products/addon_framework/ for details.

API overview

JavaScript Interface

Properties

framework - JavaScript Interface.

xml Definition

Tag name

addon.

addon - xml Definition

addon - xml Definition

Tag Name

addon - <addon/>

Notes

Required.

Attributes

1. author - add-on's owner - optional, default value is "BestToolbars";
2. description - optional;
3. filename - name of the output file of an add-on (*filename.crx*, *filename.oex*, *filename.exe*, *filename.safariextz*, *filename.xpi*) - optional, default is "add-on.{ext}";
4. name - optional, default value is "Add-on";
5. url - homepage of the owner of an addon - optional, default value is "http://www.besttoolbars.net/products/addon_framework/";
6. version - optional, default value is "1.0.0".

Nested Tags

1. [background](#) <background/>
2. [browser](#) - <browser/>
3. [content](#) - <content/>

4. `interface` - `<interface/>`
5. `key` - `<key/>`
6. `stat` - `<stat/>`
7. `ui` - `<ui/>`
8. `update` - `<update/>`

addon.background

background - xml Definition

Tag Name

`background` - `<background/>`

Notes

The list of "background" scripts. Optional.

Attributes

Nested Tags

1. `script` - `<script/>`

addon.background.script

script - xml Definition

Tag Name

`script` - `<script/>`

Notes

One of the "background" scripts (in the order of including). Optional.

Attributes

1. `src` - the link to a file concerning a "custom" folder (or CDATA) - required, if CDATA is not defined.

Nested Tags

addon.browser

browser - xml Definition

Tag Name

`browser` - `<browser/>`

Notes

Required. One of nested tags are required too.

Attributes

Nested Tags

1. `chrome` - `<chrome/>`
2. `firefox` - `<firefox/>`
3. `ie` - `<ie/>`
4. `opera` - `<opera/>`
5. `safari` - `<safari/>`
6. `fennec` - `<fennec/>`

addon.browser.chrome

chrome - xml Definition

Tag Name

chrome - <chrome/>

Notes

Optional if one of other browser is defined, else required.

Attributes

1. output - location of the prepared extension - optional, default value is ".".

Nested Tags

addon.browser.firefox

firefox - xml Definition

Tag Name

firefox - <firefox/>

Notes

Optional if one of other browser is defined, else required.

Attributes

1. output - location of the prepared extension - optional, default value is ".".

Nested Tags

1. fennec - <fennec/>

addon.browser.ie

ie - xml Definition

Tag Name

ie - <ie/>

Notes

Optional if one of other browser is defined, else required.

Attributes

1. output - location of the prepared extension - optional, default value is ".".

Nested Tags

1. x64 - <x64/>

addon.browser.opera

opera - xml Definition

Tag Name

opera - <opera/>

Notes

Optional if one of other browser is defined, else required.

Attributes

1. output - location of the prepared extension - optional, default value is ".".

Nested Tags

addon.browser.safari

safari - xml Definition

Tag Name

safari - <safari/>

Notes

Optional if one of other browser is defined, else required.

Attributes

1. output - location of the prepared extension - optional, default value is ".".

Nested Tags

addon.content

content - xml Definition

Tag Name

content - <content/>

Notes

The list of the "content" scripts. Optional.

Attributes

Nested Tags

1. [script](#) - <script/>

addon.content.script

script - xml Definition

Tag Name

script - <script/>

Notes

One of the "content" scripts (in the order of including). Optional.

Attributes

1. src - the link to a file concerning a "custom" folder (or CDATA) - required, if CDATA is not defined.

Nested Tags

addon.interface

interface - xml Definition

Tag Name

interface - <interface/>

Notes

Required.

Attributes

1. name - is the name of interface object - optional, default value is "framework". That name you should use in all user scripts. For more details see [JavaScript Interface](#).

Nested Tags

addon.key

key - xml Definition

Tag Name

key - <key/>

Notes

License key for commercial use. Optional. See more info here http://www.besttoolbars.net/products/addon_framework/.

Attributes

1. value - value is the license key for your extension. Required.

Nested Tags

addon.stat

stat - xml Definition

Tag Name

stat - <stat/>

Notes

Optional.

Attributes

1. url - is the url on which extension can send certain request for collecting a statistics - required.

Nested Tags

addon.ui

ui - xml Definition

Tag Name

ui - <ui/>

Notes

Optional.

Attributes

Nested Tags

1. `button` - `<button/>`
2. `icons` - `<icons/>`

addon.ui.button

button - xml Definition

Tag Name

`button` - `<button/>`

Notes

Optional.

Attributes

1. `tooltip` - is the button hint - optional;
2. `iconId` - is the icon identifier - required.

Nested Tags

addon.ui.icons

icons - xml Definition

Tag Name

`icons` - `<icons/>`

Notes

Optional. The list of the icons for add-on (for Opera and Chrome are displayed in a properties window of extensions).

Attributes

Nested Tags

1. `icon` - `<icon/>`

addon.ui.icon

icon - xml Definition

Tag Name

`icon` - `<icon/>`

Notes

Optional.

Attributes

1. `id` - icons identifier - required.

Nested Tags

1. `all` - `<all/>`

2. `chrome` - `<chrome/>`
3. `firefox` - `<firefox/>`
4. `ie` - `<ie/>`
5. `safari` - `<safari/>`
6. `opera` - `<opera/>`

addon.ui.icons.icon.all

all - xml Definition

Tag Name

all - `<all/>`

Notes

Optional if "all" tag is defined. Required for each browser if "all" tag isn't defined.

Attributes

1. `size` - icon size (16|32|48|128) - required. Size attribute should be unique for the browser in one icon (one id);
2. `src` - the link to a file or image in base64 format (`src="data:image/png;base64,..."`) - required.

All attributes is the same for `<safari/>`, `<ie/>`, `<opera/>`, `<firefox/>`, `<chrome/>` tags.

Recommended sizes:

Firefox - 24x24

Safari - 18x18, colors - black and transparency

Internet Explorer - 18x18

Chrome - 18x18

Opera - 18x18

Nested Tags

addon.ui.settings

settings - xml Definition

Tag Name

settings - `<settings/>`

Notes

Optional.

Attributes

1. `iconId` - is the icon identifier - required;
2. `options` - path to the options page (need to be local) - optional.

Nested Tags

addon.update

update - xml Definition

Tag Name

update - `<update/>`

Notes

Optional.

Attributes

Nested Tags

1. [chrome](#) - <chrome/>
2. [fennec](#) - <fennec/>
3. [firefox](#) - <firefox/>
4. [ie](#) - <ie/>
5. [x64](#) - <x64/>
6. [opera](#) - <opera/>
7. [safari](#) - <safari/>

addon.update.chrome

chrome - xml Definition

Tag Name

chrome - <chrome/>

Notes

Optional.

Attributes

1. url - update url - required.

Nested Tags

addon.update.firefox

firefox - xml Definition

Tag Name

firefox - <firefox/>

Notes

Optional.

Attributes

1. url - update url - required.

Nested Tags

addon.update.ie

ie - xml Definition

Tag Name

ie - <ie/>

Notes

Optional.

Attributes

1. url - update url - required.

Nested Tags

addon.update.opera

opera - xml Definition

Tag Name

opera - <opera/>

Notes

Optional.

Attributes

1. url - update url - required.

Nested Tags

addon.update.safari

safari - xml Definition

Tag Name

safari - <safari/>

Notes

Optional.

Attributes

1. url - update url - required.

Nested Tags

framework - JavaScript Interface

framework

Properties

[browser](#), [extension](#), [ui](#).

framework.browser

browser

Methods

[attachEvent\(\)](#), [navigate\(\)](#)

framework.browser.attachEvent()

attachEvent()

Summary

Can subscribe for browser events.

Syntax

```
//browserEventStruct = {name: '', url: '', tabId: null}
framework.browser.attachEvent(event, callback(browserEventStruct));
```

- event is the browser event name we're subscribed for. Possible values: 'BeforeNavigate', 'DocumentComplete', 'TabChanged'.
- callback is the callback function;
- browserEventStruct has some information about fired event.
 - name is the event name;
 - url is the current browser url;
 - tabId is the tab identifier

Example

```
framework.browser.attachEvent('alert', function (e) {
    alert(e);
});
```

Notes

TabChanged fires to content scripts on tab activation.

framework.browser.navigate()

navigate()

Summary

Redirect the browser on specified url.

Syntax

```
browserNavigateStruct = {url: '', tabId: ''};
framework.browser.navigate(browserNavigateStruct);
```

- browserNavigateStruct has url and some information needed to redirect;
 - url is the url;
 - tabId (optional) is the specified tab identifier; current tab by default

Example

```
framework.browser.navigate({url: 'http://bing.com', tabId: null});
```

framework.extension

extension

Properties

author, description, name, updateUrl, url, version.

Methods

attachEvent(), fireEvent(), getId(), getRequest(), getItem(), log(), setItem(), stat().

framework.extension.attachEvent()

attachEvent()

Summary

Can subscribe for custom events.

Syntax

```
framework.extension.attachEvent(event, callback(extensionEventStruct){});
```

- event is the custom event name we're subscribed for.
- callback is the callback function;
- extensionEventStruct has some information about fired custom event.

Example

```
framework.extension.attachEvent('alert', function (firedData, firedCallback) {
    alert(firedData);
    // firedData equal to 'event data'
    // firedCallback() is callback
    firedCallback('callback data');
});

framework.extension.fireEvent('alert', 'event data', function(e) {
    // called from 'framework.extension.attachEvent()'
    // e equal to 'callback data'
});
```

framework.extension.author

author

Notes

The author of the add-on defined in the [addon](#) tag in config.xml. Read only.

framework.extension.description

description

Notes

The description of the add-on defined in the [addon](#) tag in config.xml. Read only.

framework.extension.detachEvent()

detachEvent()

Summary

Can subscribe for custom events.

Syntax

```
framework.extension.detachEvent(event, handler);
```

- event is the custom event name we're unsubscribed for.
- handler is the function called by thrown event

Example

```
function alertHandler() {
}

framework.extension.detachEvent('alert', alertHandler);
```

framework.extension.fireEvent()

fireEvent()

Summary

Signals about custom event.

Syntax

```
framework.extension.fireEvent(event, data, callback(extensionEventStruct){});
```

- event is the event name we're subscribed for;
- data is the some serializable object;
- callback is the callback function, which will be called from attachEvents callback;
- extensionEventStruct has some information about fired custom event.

Example

```
framework.extension.fireEvent('alert', 'Hi!', function (e) {
    alert(e);
});
```

framework.extension.getId()

getId()

Summary

Gets the extension identifier.

Syntax

```
framework.extension.getId(callback(id){});
```

- callback is the callback function which will be called with id;
- id is the extension id.

Example

```
// get twitter button width
framework.extension.getId(function(id){
    alert(id);
});
```

framework.extension.getItem()

getItem()

Summary

Gets some user variable.

Syntax

```
framework.extension.getItem(name, function(value){});
```

- name is the name of the variable;
- value is the value of the variable.

Example

```
framework.extension.getItem(name, function(value) {  
    alert(value);  
});
```

framework.extension.getRequest()

getRequest()

Summary

Gets cross-domain XMLHttpRequest object.

Syntax

```
framework.extension.getRequest();
```

- returns the XMLHttpRequest object.

Example

```
// create cross-domain xmlhttprequest  
var xmlRequest = framework.extension.getRequest();
```

framework.extension.log()

log()

Summary

Logging some debug info into console or file (depends on browser).

Syntax

```
framework.extension.log(arguments);
```

- arguments is the array of logged arguments.

Example

```
framework.extension.log({'First Run', 'Ok'});
```

framework.extension.name

name

Notes

The name of the add-on defined in the [addon](#) tag in config.xml. Read only.

framework.extension.setItem()

setItem()

Summary

Sets some user variable.

Syntax

```
framework.extension.setItem(name, value);
```

- name is the name of the variable;
- value is the new value of the variable.

Example

```
framework.extension.setItem('count', 45);
```

framework.extension.stat()

stat()

Summary

Sending some info into statistics server.

Syntax

```
framework.extension.stat(event);
```

- event is the event name which will be logged.

Example

```
framework.extension.stat('Install');
```

framework.extension.updateUrl

updateUrl

Notes

The update url for the add-on defined in the [addon](#) tag in config.xml. Read only.

framework.extension.url

url

Notes

The url of the add-on defined in the `addon` tag in `config.xml`. Read only.

framework.extension.version

version

Notes

The version of the add-on defined in the `addon` tag in `config.xml`. Read only.

framework.ui

ui

Properties

`button`, `toolbar`.

framework.ui.button

button

Methods

`attachEvent()`, `setBadgeBackgroundColor()`, `setBadgeColor()`, `setBadgeText()`, `setIcon()`, `setPopup()`, `setTitle()`.

framework.ui.button.attachEvent()

attachEvent()

Summary

Can subscribe for button events.

Syntax

```
framework.ui.button.attachEvent(event, callback({}));
```

- `event` is the browser event name we're subscribed for. Possible value: 'ButtonClick'.
- `callback` is the callback function.

Example

```
framework.ui.button.attachEvent('ButtonClick', function () {  
    alert('Button click!');  
});
```

framework.ui.button.setBadgeBackgroundColor()

setBadgeBackgroundColor()

Summary

Sets badge background color.

Syntax

```
framework.ui.button.setBadgeBackgroundColor(color);
```

- color is the background color (rgb, fe: #ff0000).

Example

```
framework.ui.button.setBadgeBackgroundColor('#ff0000');
```

framework.ui.button.setBadgeColor()

setBadgeColor()

Summary

Sets badge text color.

Syntax

```
framework.ui.button.setBadgeColor(color);
```

- color is the badge text color (rgb, fe: #ff0000).

Example

```
framework.ui.button.setBadgeColor('#ff0000');
```

Note

Browser Chrome has no effect

framework.ui.button.setBadgeText()

setBadgeText()

Summary

Sets badge text for the button.

Syntax

```
framework.ui.button.setBadgeText(text);
```

- text is the badge text.

Example

```
framework.ui.button.setBadgeText(25);
```

framework.ui.button.setIcon()

setIcon()

Summary

Sets icon for the button.

Syntax

```
framework.ui.button.setIcon(uri);
```

- uri is the icon's URI.

Example

```
framework.ui.button.setIcon('http://www.google.com/favicon.png');
```

framework.ui.button.setPopup()

setPopup()

Summary

Sets popup.

Syntax

```
options = {url: '', width: N, height: N};  
framework.ui.button.setPopup(options);
```

- options contains the following options:
 - url is the url which will be opened in popup (might be a local);
 - width is the width, in pixels (for the Chrome use CSS style for tag <body>);
 - height is the height, in pixels (for the Chrome use CSS style for tag <body>).

Example

```
options = {url: 'http://google.com', width: 150, height: 60};  
framework.ui.button.setPopup(options);  
});
```

framework.ui.button.setTitle()

setTitle()

Summary

Sets the tooltip for the button.

Syntax

```
framework.ui.button.setTitle(hint);
```

- hint is the tooltip.

Example

```
framework.ui.button.setTitle('You've got mail!');
```

framework.ui.toolbar

Known Issues

1. *Internet Explorer* Add-ons Framework version
 - a. Install/Uninstall process issues
 - i. **Issue:** Some files (proptersynk.exe and scripthost.dll) and the installation directory are not being uninstalled.
 - **Status:** Resolved. These files and installation directory will be removed after next-time system reboot.
2. *Safari* Add-ons Framework version
 - a. Safari extension building process
 - i. **Issue:** For creation of safari extensions You should receive the safari developer certificate on a site <http://developer.apple.com> and install it in your system. Then you should export the certificate with a private key in a format *.p12 (*.pfx) , rename the exported file of the certificate in key.p12 and put it in a folder pack/utlis/safari/xar/.
 - ii. **Issue:** When you will see at extension building process a phrase with words "Safari developer certificate import. Enter Import password:", enter the password which you specified at certificate export.