Hack to Fix GNOME Shell Stylesheet Problems

Finnbarr P. Murphy

(fpm@fpmurphy.com)

At the present time the GNOME Shell is badly broken (See GNOME Bugzilla 650971 and 642876) with respect to its handling of GNOME Shell extension stylesheets. The issue was mainly exposed by the *user-theme* extension developed by John Stowers of *gnome-tweak-tool* fame.

The history of how this problem arose and why it still exists is interesting. In January 2011, before there was wide experience of extensions in the GNOME Shell, a leading GNOME Shell developer decided to create a GNOME Shell extensions repository and a GNOME Shell Extensions webpage. The *user-theme* extension was added to this repository in mid-March 2011 by John Stowers.

Unfortunately, the GNOME Shell Extensions webpage gives an appearance of legitimacy to the extensions included in this repository by listing them on the webpage. The assumption that a normal person would take away from reading the webpage is that the listed extensions were supported, fully tested and qualified. As a result packagers on distributions such as Fedora blessed these extensions by either including these extensions in their default install of the GNOME shell or by making them available for installation via the distribution's package manager as is the case for Fedora. Worse, these extensions are packaged so that they are installed in the global extension area, i.e. */usr/share/gnome-shell/extensions*. This led to the widespread installation and use of the *user-theme* extension by users and, as more GNOME Shell extensions with stylesheets became available, the emergence of reports of problems with styling in the GNOME Shell.

So why has the problem not been fixed if the GNOME Shell developers have been aware of the problem since at least mid-February? An interesting question to which there is no good answer. Inertia? Not invented here syndrome? Who knows! But, I believe, it is indicative of how things work within the GNOME Shell development community. If you are not a member of the inner circle of developers, you and your contributions are effectively ignored. Yes, just like our politicians and public servants, they appear to just pay lip service to inclusiveness, listening to their userbase and all that sort of thing.

A decent fix is available for this problem but has not yet been committed in spite of several code reviews and requests to have it submitted. I was one of the reviewers and testers of the proposed fix. John Stowers was another. We both have skin in the game as they say because we both have developed theme selectors. However, it would appear that we are not sufficiently blessed to review and test the proposed fixes.

Why should users continue to have a poor experience with the GNOME shell when they install one or more extensions which have stylesheets? This is just plain stupidity on the part of the GNOME Shell developers and the so-called marketing people associated with the GNOME Shell. As a temporary fix for the problem, I have decided to publish a replacement copy of the source file for the component responsible for this problem, i.e. *extensionSystem.js*. This essentially contains the fix proposed by Sardem FF7 in the *642876* bug report. While this fix is not optimum, it does fix the styling issues currently seen in the GNOME Shell.

Here is the replacement file:

```
/* -*- mode: js2; js2-basic-offset: 4; indent-tabs-mode: nil -*- */
```



```
const GLib = imports.gi.GLib;
const Gio = imports.gi.Gio;
const St = imports.gi.St;
const Shell = imports.gi.Shell;
const Config = imports.misc.config;
const ExtensionState = {
    ENABLED: 1,
    DISABLED: 2,
    ERROR: 3,
    OUT_OF_DATE: 4
};
const ExtensionType = {
    SYSTEM: 1,
    PER_USER: 2
};
// Maps uuid -> metadata object
const extensionMeta = {};
// Maps uuid -> importer object (extension directory tree)
const extensions = {};
// Array of uuids
var disabledExtensions;
// GFile for user extensions
var userExtensionsDir = null;
const _stylesheets = [];
/**
 * versionCheck:
 * @required: an array of versions we're compatible with
 * @current: the version we have
 *
 * Check if a component is compatible for an extension.
 * @required is an array, and at least one version must match.
 * @current must be in the format <major>.<minor>.<point>.<micro>
 * <micro> is always ignored
 * <point> is ignored if <minor> is even (so you can target the
 * whole stable release)
 * <minor> and <major> must match
 * Each target version must be at least <major> and <minor>
 */
function versionCheck(required, current) {
    let currentArray = current.split('.');
    let major = currentArray[0];
    let minor = currentArray[1];
    let point = currentArray[2];
    for (let i = 0; i < required.length; i++) {
        let requiredArray = required[i].split('.');
        if (requiredArray[0] == major &amp;amp;&amp;amp;
            requiredArray[1] == minor &amp;amp;&amp;amp;
            (requiredArray[2] == point ||
             (requiredArray[2] == undefined &amp;amp;&amp;amp; parseInt(minor) % 2 == 0)))
            return true;
    }
    return false;
}
function loadExtension(dir, enabled, type) {
    let info;
    let baseErrorString = 'While loading extension from "' + dir.get_parse_name() + '": ';
    let metadataFile = dir.get_child('metadata.json');
    if (!metadataFile.query_exists(null)) {
        global.logError(baseErrorString + 'Missing metadata.json');
        return;
    }
    let metadataContents;
    try {
        metadataContents = Shell.get_file_contents_utf8_sync(metadataFile.get_path());
    } catch (e) {
        global.logError(baseErrorString + 'Failed to load metadata.json: ' + e);
        return;
    }
```



```
    let meta;
    try {
        meta = JSON.parse(metadataContents);
    } catch (e) {
        global.logError(baseErrorString + 'Failed to parse metadata.json: ' + e);
        return;
    }
    let requiredProperties = ['uuid', 'name', 'description', 'shell-version'];
    for (let i = 0; i < requiredProperties.length; i++) {
        let prop = requiredProperties[i];
        if (!meta[prop]) {
            global.logError(baseErrorString + 'missing "' + prop + '" property in metadata.
json');
            return;
        }
    }
   if (extensions[meta.uuid] != undefined) {
        global.logError(baseErrorString + "extension already loaded");
        return;
    }
    // Encourage people to add this
    if (!meta['url']) {
        global.log(baseErrorString + 'Warning: Missing "url" property in metadata.json');
    }
    let base = dir.get_basename();
    if (base != meta.uuid) {
        global.logError(baseErrorString + 'uuid "' + meta.uuid + '" from metadata.json doe
s not match directory name "' + base + '"');
        return;
    }
    if (!versionCheck(meta['shell-version'], Config.PACKAGE_VERSION) ||
        (meta['js-version'] &amp;amp;&amp;amp; !versionCheck(meta['js-version'], Config.GJS_VERSIO
N))) {
        global.logError(baseErrorString + 'extension is not compatible with current GNOME
Shell and/or GJS version');
        return;
    }
    extensionMeta[meta.uuid] = meta;
    extensionMeta[meta.uuid].type = type;
    extensionMeta[meta.uuid].path = dir.get_path();
    if (!enabled) {
        extensionMeta[meta.uuid].state = ExtensionState.DISABLED;
        return;
    }
   // Default to error, we set success as the last step
    extensionMeta[meta.uuid].state = ExtensionState.ERROR;
    let extensionJs = dir.get_child('extension.js');
    if (!extensionJs.query_exists(null)) {
        global.logError(baseErrorString + 'Missing extension.js');
        return;
    }
    let stylesheetPath = null;
    let themeContext = St.ThemeContext.get_for_stage(global.stage);
    let theme = themeContext.get_theme();
    let stylesheetFile = dir.get_child('stylesheet.css');
    if (stylesheetFile.query_exists(null)) {
        stylesheetPath = stylesheetFile.get_path();
        try {
            theme.load_stylesheet(stylesheetPath);
        } catch (e) {
            global.logError(baseErrorString + 'Stylesheet parse error: ' + e);
            return;
        }
        _stylesheets.push(stylesheetPath);
    }
    let extensionModule;
    try {
        global.add_extension_importer('imports.ui.extensionSystem.extensions', meta.uuid,
```



```
dir.get_path());
        extensionModule = extensions[meta.uuid].extension;
    } catch (e) {
        if (stylesheetPath != null) {
            theme.unload_stylesheet(stylesheetPath);
            _stylesheets.pop();
        }
        global.logError(baseErrorString + e);
        return;
    }
    if (!extensionModule.main) {
        global.logError(baseErrorString + 'missing \'main\' function');
        return;
    }
    try {
        extensionModule.main(meta);
    } catch (e) {
        if (stylesheetPath != null)
            theme.unload_stylesheet(stylesheetPath);
        global.logError(baseErrorString + 'Failed to evaluate main function:' + e);
        return;
    }
    extensionMeta[meta.uuid].state = ExtensionState.ENABLED;
    global.log('Loaded extension ' + meta.uuid);
}
function init() {
    let userExtensionsPath = GLib.build_filenamev([global.userdatadir, 'extensions']);
    userExtensionsDir = Gio.file_new_for_path(userExtensionsPath);
    try {
        userExtensionsDir.make_directory_with_parents(null);
    } catch (e) {
        global.logError('' + e);
    }
    disabledExtensions = global.settings.get_strv('disabled-extensions', -1);
}
function _loadExtensionsIn(dir, type) {
    let fileEnum;
    let file, info;
    try {
        fileEnum = dir.enumerate_children('standard::*', Gio.FileQueryInfoFlags.NONE, null)
;
    } catch (e) {
        global.logError('' + e);
       return;
    }
    while ((info = fileEnum.next_file(null)) != null) {
        let fileType = info.get_file_type();
        if (fileType != Gio.FileType.DIRECTORY)
            continue;
        let name = info.get_name();
        let enabled = disabledExtensions.indexOf(name) < 0;
        let child = dir.get_child(name);
        loadExtension(child, enabled, type);
    }
    fileEnum.close(null);
}
function loadExtensions() {
    _loadExtensionsIn(userExtensionsDir, ExtensionType.PER_USER);
    let systemDataDirs = GLib.get_system_data_dirs();
    for (let i = 0; i < systemDataDirs.length; i++) {
        let dirPath = systemDataDirs[i] + '/gnome-shell/extensions';
        let dir = Gio.file_new_for_path(dirPath);
        if (dir.query_exists(null))
            _loadExtensionsIn(dir, ExtensionType.SYSTEM);
    }
}
function loadExtensionsStylesheets() {
    let themeContext = St.ThemeContext.get_for_stage(global.stage);
```



```
    let theme = themeContext.get_theme();
    for ( var i = 0 ; i < _stylesheets.length ; ++i ) {
        try {
            theme.load_stylesheet(_stylesheets[i]);
        } catch (e) {
            global.logError(baseErrorString + 'Stylesheet parse error: ' + e);
        }
    }
}
```

Here are the diffs between the two files:

```
$ diff extensionSystem.js.new extensionSystem.js.org
31,32d30
< const _stylesheets = [];
<
139d136
< stylesheetPath = stylesheetFile.get_path();
141c138
< theme.load_stylesheet(stylesheetPath);
---
>             theme.load_stylesheet(stylesheetFile.get_path());
146d142
< _stylesheets.push(stylesheetPath);
154c150
< if (stylesheetPath != null) {
---
>         if (stylesheetPath != null)
156,157d151
< _stylesheets.pop();
< }
221,233d214
<
<
< function loadExtensionsStylesheets() {
< let themeContext = St.ThemeContext.get_for_stage(global.stage);
< let theme = themeContext.get_theme();
< for ( var i = 0 ; i < _stylesheets.length ; ++i ) {
< try {
< theme.load_stylesheet(_stylesheets[i]);
< } catch (e) {
< global.logError(baseErrorString + 'Stylesheet parse error: ' + e);
< }
< }
< }
```

To install the replacement file, save a copy of your existing */usr/share/gnome-shell/js/ui/extensionSystem.js*, replace the original *extensionSystem.js* with the downloaded version, and restart your GNOME Shell.

You also need to modify */usr/share/gnome-shell/js/ui/main.js* to add the following highlighted line to it:

```
function loadTheme() {
    let themeContext = St.ThemeContext.get_for_stage (global.stage);
    let cssStylesheet = _defaultCssStylesheet;
    if (_cssStylesheet != null)
        cssStylesheet = _cssStylesheet;
    let theme = new St.Theme ({ application_stylesheet: cssStylesheet });
    themeContext.set_theme (theme);
    ExtensionSystem.loadExtensionsStylesheets();
```



```
}
```

When the final fix for this problem is available, you should replace the hacked *extensionSystem.js* with the saved *extensionSystem.js* if you are not downloading a whole new version of the GNOME Shell. You do not want two competing mechanisms for reloading stylesheets in place.

[UPDATE May 14th 2011] I decided to see if I could create a GNOME Shell extension that would provide the same functionality as manually editing *extensionSystem.js* and *main.js* using monkey patching. Here is what I came up with. It seems to work.

```
const St = imports.gi.St;
const GLib = imports.gi.GLib;
const Gio = imports.gi.Gio;
const Shell = imports.gi.Shell;
const Main = imports.ui.main;
const ExtensionSystem = imports.ui.extensionSystem;
const Config = imports.misc.config;
const _stylesheets = [];
function main() {
    // first add any stylesheets loaded prior to this extension to _stylesheets
    let userExtensionsPath = GLib.build_filenamev([global.userdatadir, 'extensions']);
    let extensions = ExtensionSystem.extensionMeta;
    for (let uuid in extensions) {
        let stylesheetFile = Gio.file_new_for_path(userExtensionsPath + '/' + uuid + '/sty
lesheet.css');
        if (stylesheetFile.query_exists(null)) {
            _stylesheets.push(stylesheetFile.get_path());
        }
    }
    // monkey patch
    ExtensionSystem.loadExtension = function(dir, enabled, type) {
        let info;
        let baseErrorString = 'While loading extension from "' + dir.get_parse_name() + '"
: ';
        let metadataFile = dir.get_child('metadata.json');
        if (!metadataFile.query_exists(null)) {
            global.logError(baseErrorString + 'Missing metadata.json');
            return;
        }
        let metadataContents;
        try {
            metadataContents = Shell.get_file_contents_utf8_sync(metadataFile.get_path());
        } catch (e) {
            global.logError(baseErrorString + 'Failed to load metadata.json: ' + e);
            return;
        }
        let meta;
        try {
            meta = JSON.parse(metadataContents);
        } catch (e) {
            global.logError(baseErrorString + 'Failed to parse metadata.json: ' + e);
            return;
        }
        let requiredProperties = ['uuid', 'name', 'description', 'shell-version'];
        for (let i = 0; i < requiredProperties.length; i++) {
            let prop = requiredProperties[i];
            if (!meta[prop]) {
                global.logError(baseErrorString + 'missing "' + prop + '" property in meta
data.json');
                return;
            }
        }
        if (ExtensionSystem.extensions[meta.uuid] != undefined) {
            global.logError(baseErrorString + "extension already loaded");
```



```
            return;
        }
        if (!meta['url']) {
            global.log(baseErrorString + 'Warning: Missing "url" property in metadata.json
');
        }
        let base = dir.get_basename();
        if (base != meta.uuid) {
            global.logError(baseErrorString + 'uuid "' + meta.uuid +
                '" from metadata.json does not match directory name "' + base + '"');
            return;
        }
        if (!ExtensionSystem.versionCheck(meta['shell-version'], Config.PACKAGE_VERSION) |
|
            (meta['js-version'] &amp;amp;&amp;amp; !ExtensionSystem.versionCheck(meta['js-version'
], Config.GJS_VERSION))) {
            global.logError(baseErrorString +
                'extension is not compatible with current GNOME Shell and/or GJS version')
;
            return;
        }
        ExtensionSystem.extensionMeta[meta.uuid] = meta;
        ExtensionSystem.extensionMeta[meta.uuid].type = type;
        ExtensionSystem.extensionMeta[meta.uuid].path = dir.get_path();
        if (!enabled) {
            ExtensionSystem.extensionMeta[meta.uuid].state = ExtensionSystem.ExtensionStat
e.DISABLED;
            return;
        }
        // Default to error, we set success as the last step
        ExtensionSystem.extensionMeta[meta.uuid].state = ExtensionSystem.ExtensionState.ER
ROR;
        let extensionJs = dir.get_child('extension.js');
        if (!extensionJs.query_exists(null)) {
            global.logError(baseErrorString + 'Missing extension.js');
            return;
        }
        let stylesheetPath = null;
        let themeContext = St.ThemeContext.get_for_stage(global.stage);
        let theme = themeContext.get_theme();
        let stylesheetFile = dir.get_child('stylesheet.css');
        if (stylesheetFile.query_exists(null)) {
            stylesheetPath = stylesheetFile.get_path();
            try {
                theme.load_stylesheet(stylesheetPath);
            } catch (e) {
                global.logError(baseErrorString + 'Stylesheet parse error: ' + e);
                return;
            }
            _stylesheets.push(stylesheetPath);
        }
        let extensionModule;
        try {
            global.add_extension_importer('imports.ui.extensionSystem.extensions', meta.uu
id, dir.get_path());
            extensionModule = ExtensionSystem.extensions[meta.uuid].extension;
        } catch (e) {
            if (stylesheetPath != null) {
                theme.unload_stylesheet(stylesheetPath);
                _stylesheets.pop();
            }
            global.logError(baseErrorString + e);
            return;
        }
        if (!extensionModule.main) {
            global.logError(baseErrorString + 'missing \'main\' function');
            return;
        }
```



```
        try {
            extensionModule.main(meta);
        } catch (e) {
            if (stylesheetPath != null) theme.unload_stylesheet(stylesheetPath);
            global.logError(baseErrorString + 'Failed to evaluate main function:' + e);
            return;
        }
        ExtensionSystem.extensionMeta[meta.uuid].state = ExtensionSystem.ExtensionState.EN
ABLED;
        global.log('MP Loaded extension ' + meta.uuid);
    };
    // monkey patch
    Main.loadTheme = function() {
        let themeContext = St.ThemeContext.get_for_stage(global.stage);
        let cssStylesheet = Main._defaultCssStylesheet;
        if (Main._cssStylesheet != null)
            cssStylesheet = Main._cssStylesheet;
        let theme = new St.Theme ({ application_stylesheet: cssStylesheet });
        themeContext.set_theme(theme);
        for ( let i = 0; i < _stylesheets.length; ++i ) {
            try {
               theme.load_stylesheet(_stylesheets[i]);
            } catch (e) {
               global.logError(baseErrorString + 'Stylesheet parse error: ' + e);
            }
        }
    };
    Main.loadTheme();
}
```

P.S. I have placed a copy of the modified *extensionSystem.js* source file on my website in the GNOME Shell Extensions download area. The extension, which I named *fixcss*, is also there.

