Intel Cloud Integrity Technology 3.0

**Navigation Bar**

# Background

Applications with a user interface typically use a navigation bar, also sometimes called a main menu, from which the user can access various features of the application.

# Architecture

## Placement

### Configurable button order

The main menu is comprised of a collection of buttons. Typically the profile and logout buttons are combined into a complex button on the far right side.

The administrator can configure a specific order for the main menu buttons to override their default positions. The setting “mtwilson.navbar.buttons” can have a comma-separated list of features as its value.

If any feature is installed but not mentioned in the “mtwilson.navbar.buttons” list, then it will appear in the alphabetically sorted “more” menu instead of in the main menu.

If any feature is mentioned in the “mtwilson.navbar.buttons” list but is not installed, it will be quietly skipped.

If any feature provides more than one button in it’s navbar main.json file, all listed buttons for that feature will be displayed in the order they are listed in the feature’s main.json file within the feature’s overall position in the menu.

### Alphabetically sorted “more” menu

Features that provide a main menu button but do not specify its position will be listed alphabetically under a “More” button. This is done in order to avoid a situation where the buttons on the navigation bar appear in a different order each time the page is loaded.

## Button Descriptor

A navbar button extension includes a main.json file describing the buttons. Each button can be described as either a label & target, which becomes a regular navbar button, or it can be described as inline html with an “li” element completely describing the navbar button, or it can be an href to an html file that contains the “li” element completely describing the navbar button. The label and target may not be mixed with inline html or href to external html.

## Content

Typically a navbar button, when clicked, will change the main view of the application to another page. The pages themselves are extensions of a separate extension point.

The other page is identified by an HTML document locator. The identifier should be set using the feature’s identifier as a prefix, for example “#com.intel.mtwilson.core.settings.tab”, and this needs to match between the href property of the button and the identifier in the HTML content of the linked page.

## Loading

The navigation bar starts loading after the user has logged in.

The index.html (main application) defines an event handler “mtwilson-core-html5:login:success.mtwilson-core-html5:navbar” that is triggered whenever the event “mtwilson-core-html5:login:success” is fired.

The event handler loads the application settings from the server and searches for the setting named “mtwilson.navbar.buttons”. If the setting is found, its value is assumed to be a comma-separated list of feature identifiers that provide buttons. If the setting is not found, the buttons will not be sorted.

To avoid a visual delay, buttons are loaded asynchronously and inserted into the navigation bar individually as they are loaded. The callback method inserts them in the sort order specified by the “mtwilson.navbar.buttons” setting.

## Home Tab

The setting “mtwilson.navbar.hometab” defines the tab that is pre-selected when the user logs in.

## Direct Links

It should be possible to hyperlink directly to a main tab. The application should update the document locator as user clicks on tabs because a page reload is not required for this.

RFC 3986 #3.5 specifies what characters are allowed in a document locator.

# Extension Point

The extension point for navigation buttons is “/mtwilson-core-html5/navbar/main.json”.

The extension point for corresponding content in the tabs is “/mtwilson-core-html5/content/main.json”

Here is an example of an extension providing just one simple button:

main.json:

{

"items": [

{ "label": "Dashboard", "target\_tab": "dashboard" }

]

}

Here is an example of an extension providing an equivalent button with inline html:

main.json:

{

"items": [

{ "html": "<li><a href='#dashboard' data-toggle='tab'>Dashboard</li>" }

]

}

Here is an example of an extension providing an equivalent button with external html:

main.json:

{

"items": [

{ "href": /html5/features/mtwilson-core-html5/mtwilson-core-html5/navbar/dashboard.html" }

]

}

dashboard.html:

<!DOCTYPE html>

<html>

<body>

<ul>

<li><a href="#dashboard" data-toggle="tab">Dashboard</li>

</ul>

</body>

</html>

Notice the external html contains a lot more than necessary; it can be extended with meta-data such as a description of the button and the author’s name:

dashboard.html:

<!DOCTYPE html>

<html>

<head>

<title>Dashboard</title>

<meta name="author" content="jbuhacoff"/>

</head>

<body>

<ul>

<li><a href="#dashboard" data-toggle="tab">Dashboard</li>

</ul>

</body>

</html>

# Security

The navigation buttons are only loaded after the user has logged in to the application. However, developers must assume that all JSON, javascript, and HTML content are public and not embed any secrets in these.

# Opens

## User interface for ordering the navigation buttons

It’s possible to add a feature where the administrator can see all available buttons (via discovery) and then drag and drop them left and right along a bar to set their position. From this activity, the setting “mtwilson.navbar.buttons” would be generated with the comma-separated list of ordered features.

## Moving navbar functionality into separate feature project

The navbar itself should be an optional feature which is added to a project like the other features.

* Create a project mtwilson-core-html5-navbar
* Main application already fires mtwilson-core-html5:login:success after user login and navbar is already initiated as a handler of this
* Navbar should check for presence of settings plugin that has already loaded the settings from the server… then instead of calling settings API directly it can just query already-loaded settings
* Navbar cannot assume that settings API or plugin is present; this is already handled with success/error callbacks and would need to be slightly revised for discovering the settings plugin instead

## Adding a “more” button with a submenu for all buttons not specifically configured

This allows the admin to keep the list of buttons shown fairly small and only the most commonly used ones would be configured there, and everything else would be shown alphabetically under a “more” or “…” button to access other features.

Even the “more” or “…” button could have a configuration for what order things should appear there, with a separator after the configured list and everything else appearing alphabetically after the separator.

## Generation of tab names

Currently tab names are hard-coded into each implementation, which may result in a conflict because these identifiers are in a global namespace.

One possibility for avoiding this issue is to use the feature identifier as a prefix to the tab name, and developers could just do this when hard-coding the tab names.

Another possibility is for the navbar to dynamically generate the tab names, using any method such as automatically prefixing the feature identifiers or just generating random identifiers for each session.

Table Design alternatives for generating tab names

|  |  |  |
| --- | --- | --- |
| **Alternative** | **+** | **-** |

|  |  |  |
| --- | --- | --- |
| Developer writes arbitrary tab names | Direct links to specific tabs are possible | Collisions likely in a global namespace |
| Developer writes arbitrary tab names prefixed with feature identifier | No collisions due to use of feature identifier as namespace (when feature identifier itself uses reverse domain name notation)  Direct links to specific tabs are possible | Longer tab identifiers for the document locator in URL, but acceptable |
| Random tab names generated by UI | No collisions | Direct links to specific tabs will not work |

# Rejected Features

## ~~Numerically sorted main menu~~

***This alternative was rejected because not likely to produce desirable results in dynamic systems and either developer or administrator likely to resort to configuring the buttons anyway.***

This feature was consideration as an alternative to requiring the administrator to define a setting “mtwilson.navbar.buttons” explicitly ordering the buttons.

When defined, the setting would override the default positions described in this alternative with the first item in the list being on the far left (0) and the last item in the list being on the far right (1).

All buttons to appear on the main menu must specify their relative decimal location in the range (0,1) with 0 on far left and 1 on far right. For example a button that should appear in the middle might be positioned at 0.5. The main menu buttons are sorted according to their stated positions.

Because new features may introduce new buttons that should appear on the main menu, they must use knowledge of the position of existing features in order to determine the desired position.

The use of decimals allows arbitrarily fine grained insertion of buttons.