This version edited: 14th September 2019

Notes for coders

Documentation for the FSC Identikit

Original development funded by the Esmée Fairbairn Foundation and the Heritage Lottery Fund

1 Contents

2 Introduction

This guide includes only the sketchiest details on the architecture of the FSC Identikit (referred to below as ‘the Identikit’). It may be enough to get you going, but if you want more information, please contact Rich Burkmar (richardb@field-studies-council.org) and more information will be provided (possibly by adding relevant information to this document).

3 General notes

The Identikit is written in JavaScript and CSS. It also relies heavily on a couple of open-source JavaScript libraries: d3.js and jQuery.js (other dependencies are listed below). D3 is used mainly to provide the animated graphics for the interactive multi-access keys. Using these standard web technologies enables us to avoid any dependencies on third-party plugins and ensures that the visualisations should work on most modern browsers.

4 Software architecture

The FSC Identikit is a software framework: if you create a new visualisation object to the specified standards, that object will be picked up and acted upon by the Identikit.

The JavaScript code is modularised and an entirely new visualisation can be created simply by copying the template visualisation module and modifying it to suit your ideas (see the section ‘Creating a new visualisation’). All the code and resources that directly comprise the Identikit architecture are to be found in the folder ‘tombio’.

The main components of the architecture are *briefly* described below.

4.1 Top-level folder

The sw.js file is the service worker which can manage caching of the resource files, e.g. images and knowledge-base files, when Identikit is deployed to websites. The manifest.json file contains some configuration information that allows Identikit deployments to be ‘installed’ to mobile devices as ‘Progressive Web Apps’.

The ‘vis.html’, ‘vism.html’ and ‘site.css’ files are template deployment files which are themselves used by the Electron Identikit knowledge-base developer’s interface, which is itself defined by the ‘electron.html’ and ‘electron.js’ files.

The ‘gulpfile.js’ file is a configuration file for the ‘gulp’ utility which can be used to minify Indentikit CSS and JS files.

4.2 Main HTML and CSS files

The ‘**visInfo.html**’ file contains the general information on the Identikit that is shown to users when they select the ‘About FSC Identikit’ option from the ‘Select a tool’ drop-down.

The ‘**tombiovis.css**’ file contains most of the CSS responsible for styling and laying out elements of the Identikit. It currently holds some CSS that is specific to individual visualisations (particularly ‘vis1’, ‘vis2’ and ‘vis3’) and which would be better placed in separate CSS files for those modules.

4.3 Main JavaScript files

The '**load.js**' JavaScript file is the top-level JavaScript in the Identikit. It's this file that web pages that implement the Identikit must include. It is responsible for loading all the Identikit JavaScript and CSS dependencies and reading the knowledge-base files.

The '**tombiovis.js**' JavaScript file contains much of the core Identikit code which is independent of the actual visualisations.

The '**score.js**' JavaScript file contains just the code responsible for scoring taxa against user character state input. This is discussed in much more detail in a separate document – 'Character scoring'.

The '**visP.js**' JavaScript file is a module that defines an object which is used as a prototype for all the templates. It contains many functions that could be of general use to coders of new visualisations, including, for example, functions for displaying and manipulating images.

The '**taxonselect.js**' file contains the code responsible for building the taxon selection control which is used by visualisations such as the vis4 ('Full taxon details') and vis3 ('Side by side comparison').

The '**keyinputTemplate.js**' file defines an interface which any taxon character input controls must match in order to work with Identikit.

The '**keyinputBasic.js**' file defines a basic taxon character input control which is not used by any visualisations in Identikit.

The '**keyinput.js**' file defines a taxon character input control that uses pqSelect controls. This is the keyinput control used by vis1 ('Two-column key'), vis2 ('Single-column key') and vis3 ('Circle-pack key') on large format implementations.

The '**keyinputOnsenUi.js**' file defines a taxon character input control that uses the Onsen UI framework. This is used by vis1 ('Two-column key'), vis2 ('Single-column key') and vis3 ('Circle-pack key') on mobile-first implementations.

The '**guiTemplate.js**' file defines an overall interface which any overall interfaces must match in order to work with Identikit. It is also a template which can be used as a starting point for any new tools added.

The '**guiLarge.js**' file defines a basic overall interface which is not used.

The '**guiLargeJqueryUI.js**' file defines an overall interface based on jQuery UI. This is used on large format implementations.

The '**guiLargeOnsenUi.js**' file defines an overall interface based on the Onsen UI framework. This is used on mobile-first implementations.

4.4 The 'common' folder

The common folder stores HTML files, and any images referenced by them, that provide help to users on elements of the Identikit which can be referenced by a number of modules, for example the user state input controls and image display tools.

4.5 The 'dependencies' folder

The dependencies folder contains all the resources associated with the software dependencies of the Identikit including:

- D3
- jQuery
- jQueryUI
- Onsen UI
- pqSelect (for creating user state input controls)
- pqGrid (used by the 'vis3' visualisation for creating side-by-side comparisons)
- galleria for handling display of images.
- Electron-pdf-winod – a modified version of the Node.js package to avoid some vulnerabilities in published package.

4.6 The 'resources' folder

The resources folder contains images required for the main Identikit code.

4.7 The 'css' folder

The css folder contains styling resources for the interface elements described in the section 'main Javascript files'.

4.8 The 'min' folder

The min folder contains the minified JS and CSS files (in sub-folders) for all the JS and CSS comprising the Identikit. These files are generate by the 'gulp' utility as defined in the 'gulpfile.js' configuration file in the

4.9 The 'vis*Name*' folders

The 'vis*Name*' folders correspond to the individual visualisations. They contain all the JS code, CSS and resources that comprise an individual visualisation. The folder 'visT' is the special template visualisation folder on which new visualisations can be based (see 'Creating a new visualisation' section).

5 Creating a new visualisation

5.1 Getting started from the template visualisation module

To create a new visualisation, carry out the following steps:

- Make a copy of the 'visTemplate' visualisation template folder and rename it to a unique shorthand name for your visualisation. Although not mandatory, it is a good idea to keep the 'vis' prefix to identify this as a visualisation module. We'll call the new module 'visExample' for the purposes of this documentation.

- Open the new folder (e.g. 'visExample') and rename the javascript code file to match your visualisation's name, e.g. 'visExample.js'.

- Optionally create a new CSS file in the same folder to match your visualisation's name, e.g. 'visExample.css'. You can put CSS specific to your visualisation in here and it will be automatically picked up.

- Edit the value of the 'visName' variable in the visualisations javascript module (e.g. 'visExample.js') to match the visualisation's name, e.g. 'visExample':

```
        var visName = "visExample";
```

- Comment out the line 'var visT = tbv.templates[visName] = Object.create(tombiovis.v.visP);', uncomment the line below that.

- Uncomment the line '//tbv.f.checkInterface(visName, tbv.templates.visTemplate, tbv.v.visualisations[visName]);' which will ensure that the interface of your new tool is checked against that defined by 'visTemplate.js'.

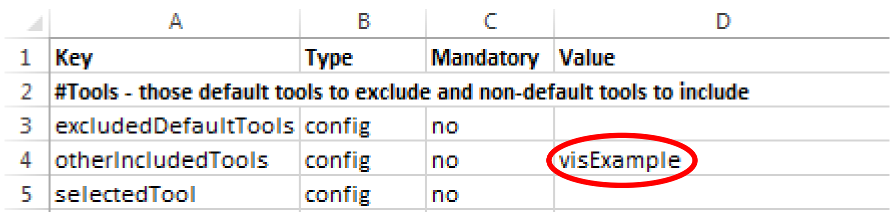
- Replace all references to 'visT' to match your visualisation, e.g. 'visExample'.

- Edit the module's metadata variables, e.g:

```
        this.metadata.title = "Example visualisation";
        this.metadata.authors = "Rich Burkmar";
        this.metadata.year = "2016";
        this.metadata.publisher = null;
        this.metadata.location = null;
        this.metadata.contact = "richardb@field-studies-council.org";
        this.metadata.version = "1.0";
```

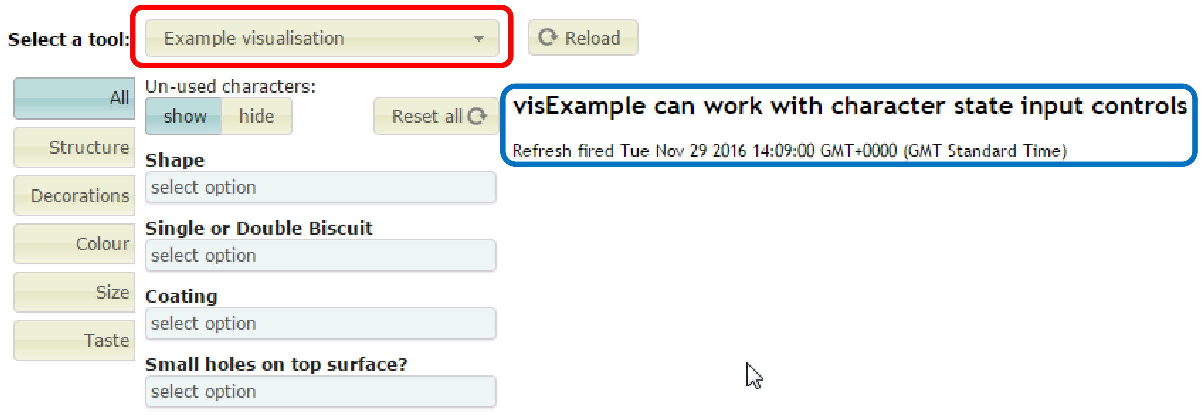
- If your visualisation needs input from the user on taxon character states, you need to link it to a keyinput control. To see how to do that, look for the following line in vis1: vis1.inputControl = tbv.gui.sharedKeyInput[keyinput];

- In whatever knowledge-base(s) you are working with, modify the value of the 'otherIncludedTools' key on the config worksheet to specify that the Identikit should include your new module, e.g. 'visExample', and regenerate the CSV files.



	A	B	C	D
1	**Key**	**Type**	**Mandatory**	**Value**
2	**#Tools - those default tools to exclude and non-default tools to include**			
3	excludedDefaultTools	config	no	
4	otherIncludedTools	config	no	visExample
5	selectedTool	config	no	

Now when you run the Identikit (refresh it if it's already running) your new visualisation will be picked up by the Identikit and presented as an option in the 'Select a tool' drop-down list (outlined in red below). The visualisation itself, at this point will do little except display the messages you can see outlined in blue.



visExample can work with character state input controls

Refresh fired Tue Nov 29 2016 14:09:00 GMT+0000 (GMT Standard Time)

5.2 Coding your visualisation

There are two main places where you can put code in your module – the 'initialise' and 'refresh' functions. The 'initialise' function is called just once by the Identikit when the visualisation is first invoked for the first time. The 'refresh' function is called whenever the user state input controls are used and when your visualisation is redisplayed after another one has been used.

You can, of course, provide and use as many helper functions as you wish.

Don't forget that the prototype of your visualisation object comes from the 'visP.js' module and so your visualisation has access to all the functionality coded in there, including that for sophisticated display of photographs, displaying knowledge-base values and so on.

Two of the current default visualisations ('vis1' and 'vis2') use SVG shapes to represent taxa and the D3.js library to manipulate them. If you want to get a feel for how that works, look at these modules.

5.3 Help files for your visualisation

You should provide a HTML help file for your visualisation. This file (or files) should be referenced by the 'helpFiles' variable in the 'initialise' function. If your visualisation uses any of the standard functionality, e.g. user state input controls, or image display, then you should also reference the help files provided for these. These are stored in the 'tombio/common' folder.

So, for example, if you provided a help file called 'visExample.html' in your module's folder *and* you also used the user state input controls, then you should set the value of 'helpFiles' to something like that shown below:

```
this.helpFiles = [
     tombiopath + "vis4Example/visExample.html",
     tombiopath + "common/stateInputHelp.html"
]
```

The Identikit will concatenate these files and display to users in a dialog when the 'About this visualisation tool' button is clicked.

Note that if you include resources in the html, such as images, then these must be located within your module folder (best within a subfolder of it). To reference these from the HTML you need to use a special token – '##tombiopath##' – in your HTML to specify the pathnames (see the example below from the 'vis2' visualisation.

```
<img src="##tombiopath##vis2/resources/one-column.png" style="float: right; width: 
300px" />
```

The Identikit will replace this token with the value of the ‘tombiopath’ variable (see documentation on deploying the Identikit to see where this is set) ensuring that you can easily move your visualisation code can be moved around easily (e.g. when it is deployed) without having to edit these files.

6 Open-source collaboration

We would welcome collaboration with other coders to improve and extend the FSC Identikit framework. That is one reason we chose GitHub as a repository for the source code.