This version edited: 29th November 2016

Notes for coders

Documentation for the Tom.bio ID Visualisation Framework

Dr Richard Burkmar
Biodiversity Project Officer
Field Studies Council
Head Office
Montford Bridge
Shrewsbury
SY4 1HW

richardb@field-studies-council.org
Tel: (01743) 852125

Tomorrow's Biodiversity Project
funded by the Esmée Fairbairn
Foundation

1 Contents

1 Contents 2
2 Introduction 3
3 General notes 3
4 Software architecture 3
4.1 Top-level HTML and CSS files 3
4.2 Top-level Javascript files 3
4.3 The ‘common’ folder 4
4.4 The ‘dependencies’ folder 4
4.5 The ‘resources’ folder 4
4.6 The ‘vis*Name*’ folders 4
5 Creating a new visualisation 5
5.1 Getting started from the template visualisation module 5
5.2 Coding your visualisation 6
5.3 Help files for your visualisation 6
6 Open-source collaboration 7

2 Introduction

The first release of this guide includes only the sketchiest details on the architecture of the Tom.bio ID Visualisation Framework (referred to below as 'the framework'). 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 framework 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

We call it a framework partly because it fits the definition of a framework in the software engineering sense of the word: if you create a new visualisation object to the specified standards, that object will be picked up and acted upon by the framework.

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 framework architecture are to be found in the folder 'tombio'.

The main components of the architecture are *briefly* described below. We hope to expand on this documentation in future releases.

4.1 Top-level HTML and CSS files

The contents of the '**import.html**' file is injected into the top level div element with the ID 'tombiod3'. Many of the interface elements are dynamically created by the template, but currently some of the top-level elements are not and are, instead, laid out in this file.

The '**visInfo.html**' file contains the general information on the entire framework that is shown to users when they select the 'About Tom.bio ID visualisations' 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 visualisation framework. 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.

4.2 Top-level JavaScript files

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

The ‘**tombiovis.js**’ JavaScript file contains most of the framework code which is independent of the actual visualisations. So, for example, all the code for creating the state input controls and handling user responses is contained in here.

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.

4.3 The ‘common’ folder

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

4.4 The ‘dependencies’ folder

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

- D3
- jQuery
- jQueryUI
- spin.js (for creating the spinner that displays whilst the framework loads)
- pqSelect (for creating user state input controls)
- pqGrid (used by the ‘vis3’ visualisation for creating side-by-side comparisons)
- jQuery Mousewheel (for handling mouse events for image handling)
- jQuery TouchPunch (for making more friendly for touch devices – needs more work)

4.5 The ‘resources’ folder

The resources folder contains images required for the top-level framework code.

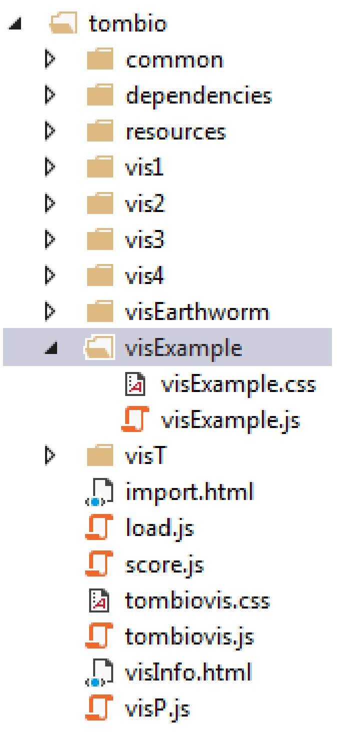
4.6 The ‘vis*Name*’ folders

The ‘vis*Name*’ folders correspond to the individual visualisations. They contain all the code 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 you carry out the following steps:


```
tombio
  common
  dependencies
  resources
  vis1
  vis2
  vis3
  vis4
  visEarthworm
  visExample
    visExample.css
    visExample.js
  visT
  import.html
  load.js
  score.js
  tombiovis.css
  tombiovis.js
  visInfo.html
  visP.js
```

- Make a copy of the 'visT' visualisation template folder in the 'tombio' 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.

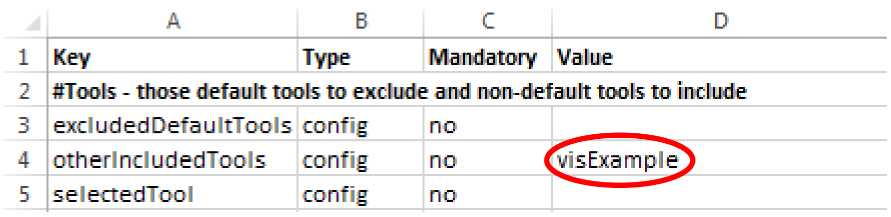
- Edit the value of the 'visName' variable in the visualisations javascript module (e.g. 'visExample.js') to match the visualisation's name, e.g. 'visExample':

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

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

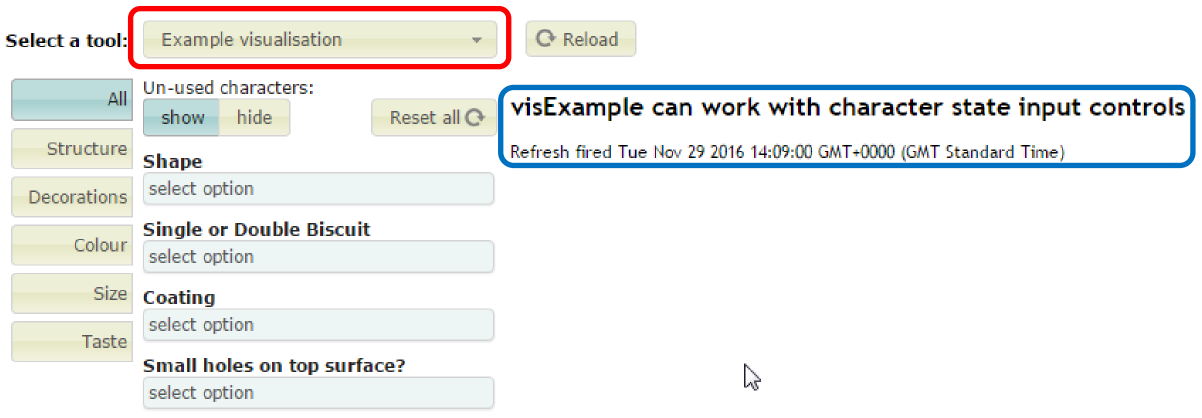
```
this.metadata.title = "Example visualisation";
this.metadata.authors = "Rich Burkmar";
this.metadata.year = "2016";
this.metadata.publisher = null;
this.metadata.location = null;
this.metadata.contact = "richardb@field-studies-council.org";
this.metadata.version = "1.0";
```

- In whatever knowledge-base(s) you are working with, modify the value of the 'otherIncludedTools' key on the config worksheet to specify that the framework 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 framework (refresh it if it's already running) your new visualisation will be picked up by the framework 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.



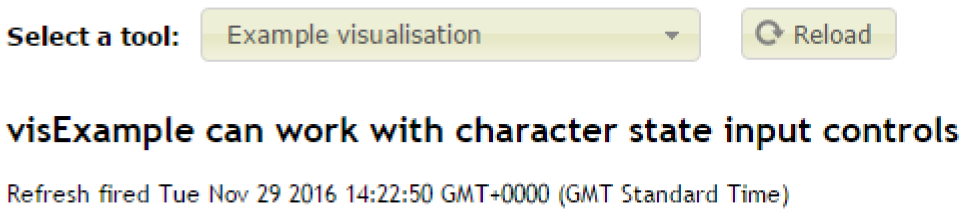
visExample can work with character state input controls

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

Notice that the new visualisation is set up to use the default state input controls provided by the framework. If your visualisation does not require state input from the user, or if you plan to provide your own state input controls, you can turn off the default state input controls by changing the value of the ‘charStateInput’ variable in the ‘Initialise’ function of your module from ‘true’ to ‘false’:

```
    this.charStateInput = false;
```

This will result in the visualisation looking like this:



visExample can work with character state input controls

Refresh fired Tue Nov 29 2016 14:22:50 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 framework 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 framework 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 framework will replace this token with the value of the ‘tombiopath’ variable (see documentation on deploying the framework 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 Tom.bio visualisation. That is one reason we chose GitHub as a repository for the source code. To be candid, I (Rich Burkmar) have limited knowledge of collaborative open-source development. So although I am convinced that GitHub is an excellent platform on which to achieve this, I am not very familiar with the actual practice of using it to realise that goal. So if you would like to contribute to the project, e.g. by contributing new visualisation modules, please feel free to communicate with me through GitHub, but please understand that I myself will have to undergo a learning curve on collaborative development through GitHub!