Advent Calendar 2015 in vanilla JavaScript

This project was entirely inspired by the Flipbook article by Hongkiat. After reading the article, downloading the project and learning all I could from it, I decided to have a go myself. My approach is far less Object-Oriented Programming (OOP), which was one of the key aspects of the project in the article.

My interpretation of the Advent Calendar also uses vanilla JavaScript and I make use of the same messages.js file containing the surprise messages. As well as using vanilla JS I have also attempted to employ the most appropriate technology for each feature. HTML for all the DOM code and CSS for all the styling; split between layout-related styles and theme styles.

Other key differences in my solution include:

- Able to use a background selected at random from a folder of image files. There is a limitation: they need to be 800x600 or 600x800 pixels.
- The windows (DIV) are resized depending on the orientation or the background image and they flow to fit the available space, using a CSS class.
- The page header is updated to include a reference to the year of the calendar.
- I chose to re-order the windows so they start in the bottom left corner and run left-to-right, then bottom-to-top.
- As a matter of course I employ the "use strict" directive to keen my code as bug-free as possible.

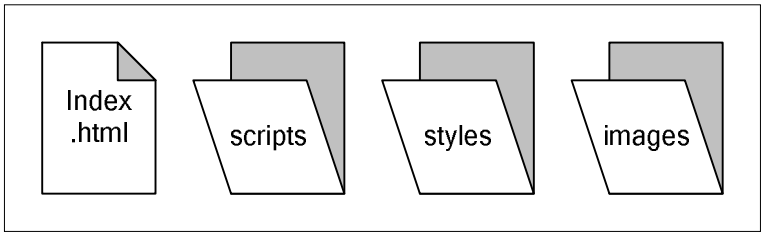
Some things I don't do:

- I do not use OOP to create and configure the doors/windows.
- The dimensions of the windows are not calculated on the fly but configured using CSS classes (main.land and main.port) once the orientation of the background image is calculated.
- There is a single 'on click' event handler for opening the windows, attached to the DIV containing all the windows, rather than attaching an event handler to up to 24 DOM objects.

This article was written to accompany the implementation of the project. Please read on how I went about it and take/leave the aspects you think worthwhile for your projects.

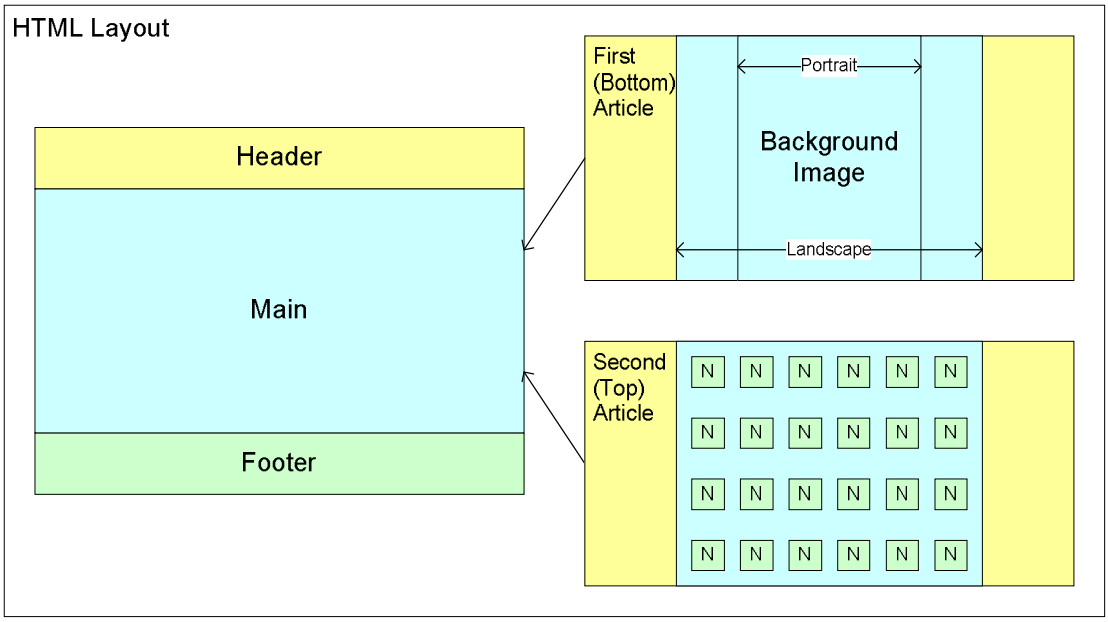
Folders and Files

In the top-level folder resides a single html file alongside three folders (images, styles and script). In my example each folder contains two files but more images could be added.



HTML

The BODY element is split into three full-width sections; header, main and footer. The header and footer contain only text but the header does get some special attention from the JavaScript later. The footer highly gives credit to the author of the original article and provides links.



The main element is invisible until the orientation of the background image is discovered and a PORTrait or LANDscape class is assigned. With the MAIN element there are two full-width articles that are presented one on top of the other. The bottom one contains the background image and has a fixed height of 600px. This means landscape images are shown full size but portrait backgrounds are reduced to a width of 450px to maintain a consistent height and aspect ratio.

The second ARTICLE contains a DIV with the class “calendar” which, in combination with the MAIN element class (port or land) with re-size to align with the background image. Inside the DIV.calendar there are 24 “window” DIVs that flow to fit the available container, with a consistent dimensions and spacing.

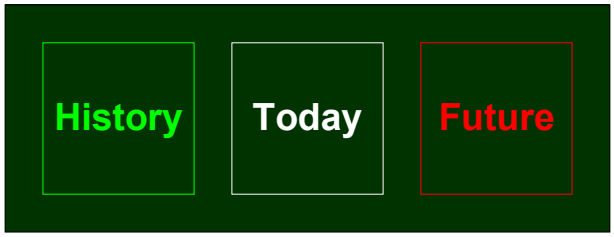
Images

There are two images, one landscape and the other portrait in orientation. Both files are itemised in the JavaScript code array (arrImages) in the calendar.js file. The number of files can be increased by adding them to the folder and including them in the array. The new files will be selected at random as the backdrop for the calendar but they do need to be 800x600 pixels or 600x800px for best results.

Styles

All styles used by the web page are contained within one of two files, as CSS classes, depending on their purpose (layout.css or theme.css.) The JavaScript code does not change any DOM node style property directly but does apply CSS classes using the className property; although the classList property can be used, where available.

Three CSS classes (today, history and future) not only colour the windows, they also help the JavaScript identify what to do with the windows when clicked. Open (today and history) windows or do nothing with (future) windows.

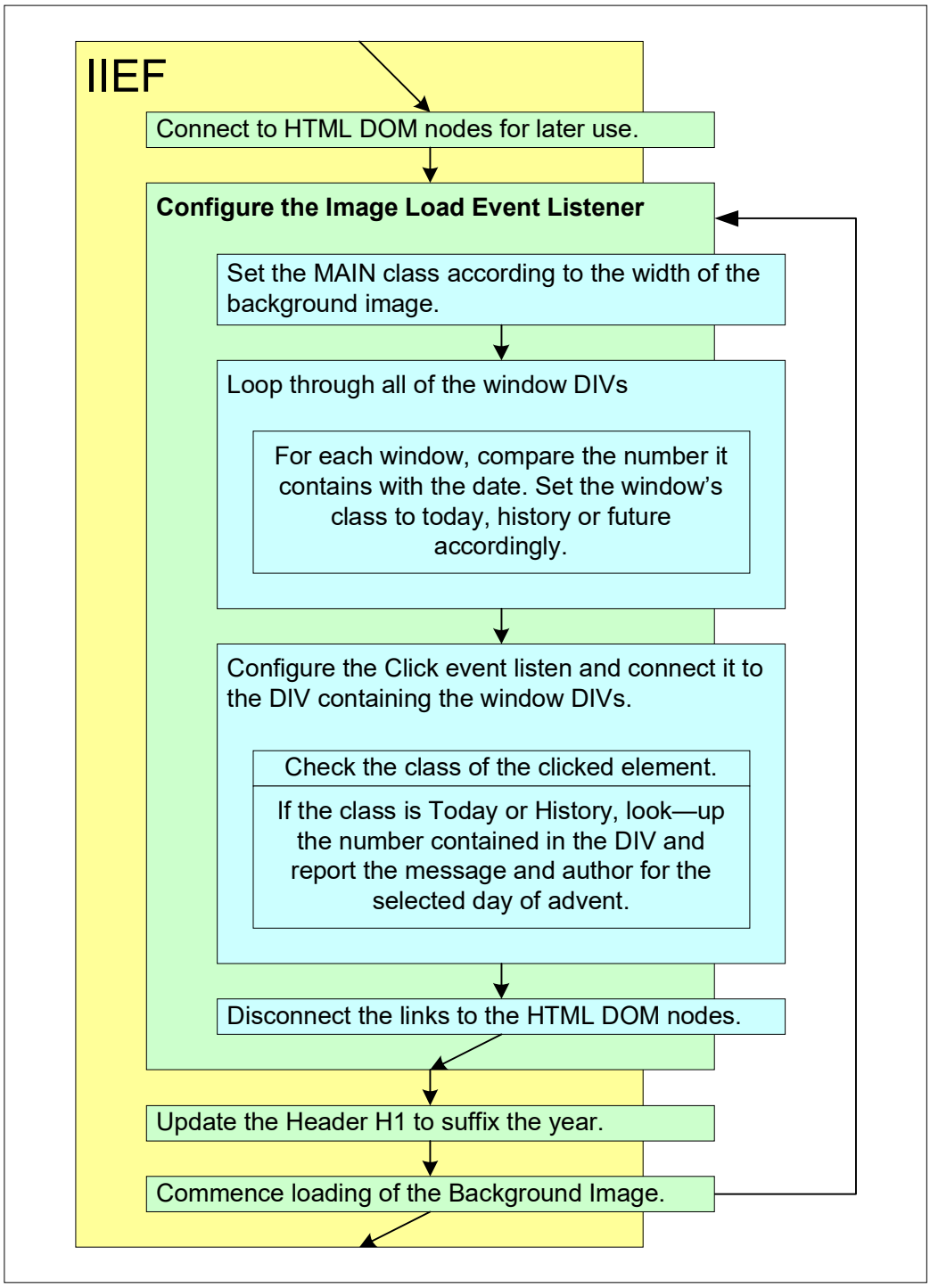


The styles that affect the theme of the page include font family, size and weight. It also includes visibility of elements along with the colour of text, background and borders.

The layout classes define the location and size of panels within the page, including padding and margins. They also include the alignment and floating of text.

Scripts

In addition to the messages.js file borrowed from the original project, there is my own version of the calendar.js file. Both files are referenced at the bottom of the HTML file; last elements of the BODY tag. All the code used in the application is implemented with three (nested and anonymous) functions.



The (outermost) all-encompassing function is an Immediately-invoked_function_expression (IIFE), which commences execution as soon as the function is defined and runs once. The primary purpose of this IIFE is to contain all the variables required and avoid contaminating global scope.

Connections between the JavaScript engine and the HTML DOM are cached; connected (using the querySelector method) at the start and released once complete.

Inside the IIFE the background image is selected, a “load” event listener prepared and the image is loaded. Once the background image has completed loading, its associated call-back function is fired that performs the following tasks.

- Detect the width of the rendered image and calculate the aspect ratio.
- Set the class of MAIN to ‘land’ or ‘port’ according to the aspect ratio. This has the effect of configuring the window DIVs and making them visible.
- Loop through all window DIVs configuring it class depending on the number of the window and the current day of the month.
    - If the window is less than today’s day, set it to ‘history’.
    - If the window is equal to today set it to ‘today’.
    - Otherwise set it to future.
- Attach the ‘click’ event handler (listener) to calendar DIV, which contains all the window DIVs, to perform the following when the mouse is clicked inside the DIV.
    - Locate the DOM node that was actually clicked.
        - In new browsers this will be event.target,
        - Older browsers (MSIE) will use the event.srcElement property.
    - Check the class on the clicked node is equal to ‘today’ or ‘history’, otherwise disregard the click event.
    - Extract the number presented in the clicked window and show the associated message for the day.