CAROUSEL TEMPLATE DEVELOPERS GUIDE

REQUIREMENTS

- BxSlider Plugin http://bxslider.com/
- Ss15-carousel-template.js
- SS15 Feature HTML Template

BASIC SETUP

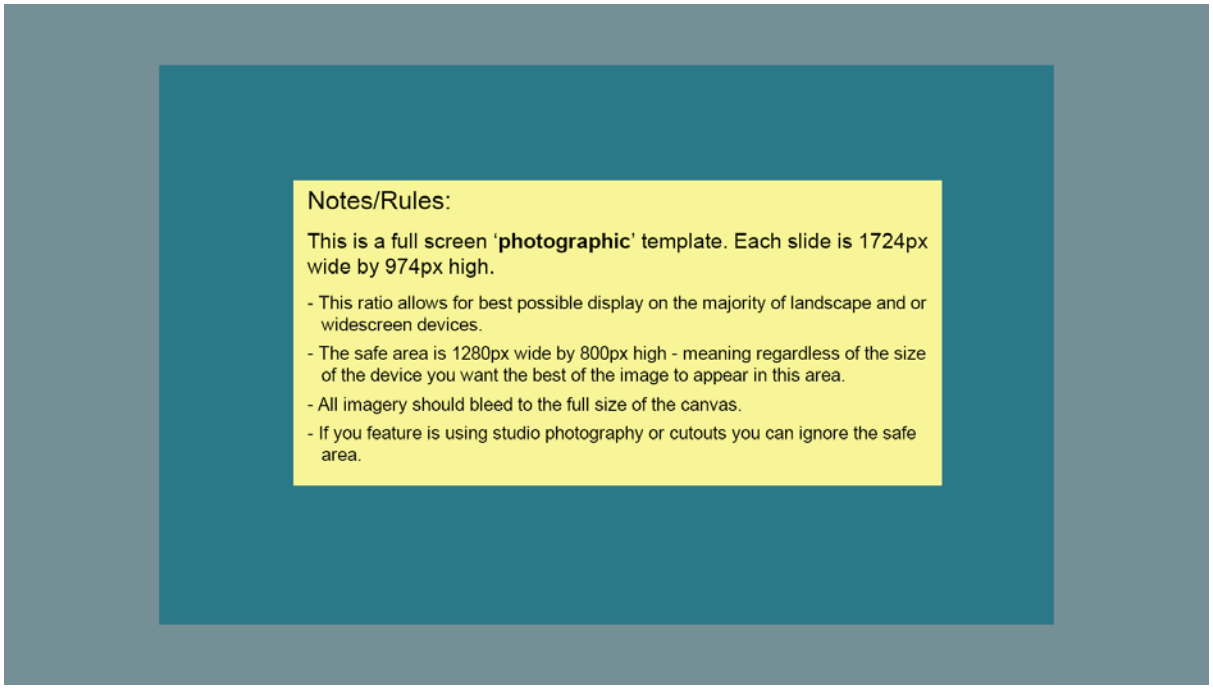
1. Download the zip from the GitHub page (https://github.com/zachtownsend/ss15-feature-template)
2. Paste the "template" folder into the features content folder. Rename it appropriately.
3. Paste the "images" folder into the features image folder. Rename it appropriately.
4. Open the CB1.htm file that you pasted in step 2 into your code editor of choice.
5. Go to script at the bottom of the file and modify the custom settings. As a minimum you should change:
    a. `cm_tag` – this is the name the coremetrics tagging uses
    b. `img_path` – this is the path to the folder you pasted in step 3.
    c. `product_codes` – you can change these later on if you wish, or you can delete them altogether and disable the product grid functionality.
6. Edit HTML and CSS to your liking.

IMAGE CONVENTIONS

IMAGE DIMENSIONS

Fullscreen background images

For fullscreen images that need to fill the viewport window, the fullscreen guide should be used.



Any essential content needs to be in the active area in the middle, and the buffer area is just there to allow for bleed on different window sizes. From the designer's point of view, a good rule of thumb is to *assume anything in the buffer area will be cropped*!

Thumbnails

For thumbnail images, export them at 400px width.

NAMING CONVENTIONS

Slide-specific prefix

For images for a specific slide, the file name should begin with '*slide**x**-*' where ***x*** is the slide number. The slide numbers are 0-based, so the very first slide would be '*slide0-*'.

Background images

The background images for each slide should be the slide-specific prefix followed by '*bg*'. E.g. *slide0-bg.jpg*

Thumbnails

The thumbnail images for each slide should be the slide-specific prefix followed by '*thumbnail*'. E.g. *slide0-thumbnail.jpg*

CSS GUIDE

CSS STRUCTURE

Normal Styles

These styles are the ones that are used in the fixed-width mode.

Fullscreen Styles

These styles are the ones that are used in the responsive full-screen mode. All fullscreen specific styles should start with `.fullscreen` because the plugin adds a class of `fullscreen` to the `body` tag.

Special Styles

These are styles for special functionality, such as the styles required for animated jpegs, or UI switching.

Background position classes

Sometimes you will need to position a fullscreen background image in a certain way (e.g. align it with the top). To do this, you can simply add a helper class to the `.slide-container` div in the HTML. Here is a list of the helper classes you can copy and paste (what they do should be fairly self-explanatory):

```
bg-top
bg-top-left
bg-top-right
bg-bottom
bg-bottom-left
bg-bottom-right
bg-left
bg-right
```

Custom Feature Styles

This is where you put your feature specific styles.

Media Queries

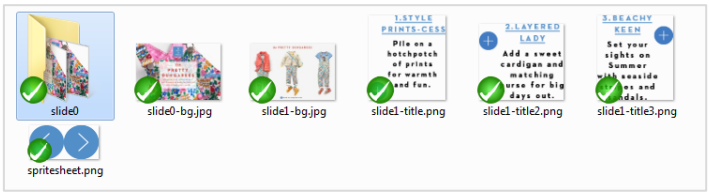
At the bottom of the CSS you can find the media queries for iPad portrait mode.

JPEG ANIMATION FUNCTIONALITY

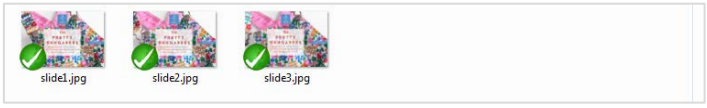
This functionality allows you to "animate" jpeg images. It adds a series of tags, one for each frame, into a container and animates the z-index to iterate through each frame. To set this up, follow these steps.

INSTRUCTIONS

1. Create a folder in the features images folder for every slide that you wish to have jpeg animation. For example, if you want jpeg animation of the intro slide, create a folder named "slide0". Put the frames into this folder, named "slide1.jpg, slide2.jpg etc.".



1 Feature root image folder showing the frame sub-folder ("slide0" for intro slide)



2 Inside the "slide0" sub-folder

2. Now, go to the HTML template file and place an empty div tag with a class of "bg-container" (i.e. `<div class="bg-container"></div>`) as the first child of the `.content-container` div for every slide that you wish a Jpeg animation to appear.

```
1015 <div class="feature-wrapper">
1016     <div id="viewport-container">
1017         <ul class="bxslider">
1018             <li id="intro" class="active-slide slide0">
1019                 <div class="slide-container"> <!--BG position goes here-->
1020
1021                     <div class="content-container">
1022                         <div class="bg-container"></div>
1023                         <div class="copy">
1024                             <h2>Lorem Ipsum</h2>
1025                             <p>Lorem Ipsum</p>
1026                             <a href="#" class="next">Lorem Ipsum <span class="r-arw"></span></a>
1027                         </div>
1028                     </div>
1029
1030                 </div>
1031             </li>
```

3. Go down to your inline JavaScript at the bottom of the page, and enable the jpeg animation

```
1105 var custom_settings = {
1106     global_settings: {
1107         cm_tag: 'SS15-TEMPLATE',
1108         img_path: '/images/magazine/features/SS15-carousel/',
1109         product_codes:
1110             [
1111                 null,
1112                 'WH793LAV|WU003PPK|AR660NUD|AV090YEL',
1113                 'WK959SPK|WH779BLU|AR654PPK',
1114                 'WG590RED|AR662BLK',
1115                 'WC114OAT|WV020GRY|WE479OYS|AR662BLK',
1116                 'WE465NAV|WH787PPK|AR654BLK|AM224YEL'
1117             ]
1118     },
1119     jpganimate_settings: {
1120         enabled: true
1121     }
1122 }
```

4. Next, define the amount of frames for each animation in the `scene` array. Each cell of the array represents the slide number, and the number you provide is the number of frames for that animation. Each slide should have its own cell in the

array, so if no animation is required for that slide, give the corresponding cell a value of `null`.

```
var custom_settings = {
    global_settings: {
        cm_tag: 'SS15-TEMPLATE',
        img_path: '/images/magazine/features/SS15-carousel/',
        product_codes:
            [
                null,
                'WH793LAV|WU003PPK|AR660NUD|AV090YEL',
                'WK959SPK|WH779BLU|AR654PPK',
                'WG590RED|AR662BLK',
                'WC114OAT|WV020GRY|WE479OYS|AR662BLK',
                'WE465NAV|WH787PPK|AR654BLK|AM224YEL'
            ]
    },
    jpganimate_settings: {
        enabled: true,
        scene: [3, null, null, null, null, null]
    }
}
```

5. If everything is correct, the jpeg should animate when the user lands on that slide.

SETTINGS

You can modify how the jpeg animation mode behaves by editing the default settings. See below for details.

Default Settings

```
jpganimate_settings: {
    enabled: false,
    container: '.bg-container',
    slide_prefix: '.slide',
    img_path: global_settings.img_path,
    speed: 300, //in ms
    repeat: -1,
    paused: true,
    yoyo: true,
    repeat_delay: 0, //in ms
    scene: [2, null, null, null, null, null]
}
```

speed

This sets the frame rate of the animation in milliseconds.

repeat

This sets how many times you want the animation to loop. `-1` sets it to an infinite loop.

repeat_delay

This sets the delay between repeats in milliseconds.

paused

This sets the default initial state to be paused. You should leave this as it is as the bx-slider will play this anyway in its call backs. If you set this to `false` then all animations will play at once (not desirable).

yoyo

If `true`, the animation will go to the end and then go in reverse back to the beginning, rather than go straight back to the first frame.

STAGGERED LAYERS FUNCTIONALITY

INSTRUCTIONS

1. First of all, go to the inline script at the bottom of the HTML template and enable the `layers` functionality

```
var custom_settings = {
    global_settings: {
        cm_tag: 'SS15-TEMPLATE',
        img_path: '/images/magazine/features/SS15-carousel/',
        product_codes:
            [
                null,
                'WH793LAV|WU003PPK|AR660NUD|AV090YEL',
                'WK959SPK|WH779BLU|AR654PPK',
                'WG590RED|AR662BLK',
                'WC114OAT|WV020GRY|WE479OYS|AR662BLK',
                'WE465NAV|WH787PPK|AR654BLK|AM224YEL'
            ]
    },
    layers_settings: {
        enabled: true
    }
}
```

2. Next, inside the slides `.content-container` div, add a class of `layer` to each element you want to stagger.

```
<li id="slide1" class="slide1">
    <div class="slide-container bg-top">
        <div class="content-container">
            <div class="copy layer">
                <p>
                    <span class="layer"><img src="/images/magazine/features/wow-now-outfits-spring-knits/quotemark-left.png" class="q
                    <img src="/images/magazine/features/wow-now-outfits-spring-knits/quotemark-right.png" class="quote-right" /></spa
                    <img class="layer" src="/images/magazine/features/wow-now-outfits-spring-knits/slide1-copy.png" alt="" />
                </p>
                <a href="~/products/outfitting.aspx?qr=WV019LPR|WG579BLK|AR657BLK|AM224NAV" class="quicklink layer retrieve">Get the
            </div>
        </div>
    </div>
</li>
```

3. The stagger animation should now happen when the user navigates to that slide.

SETTINGS

Default settings

```
layers_settings: {
    enabled: false,
    layer_class: 'layer',
    duration: 1,
    delay: 0.4
}
```

duration

Total duration of each layer's animation

delay

The delay between the beginning of each layer's animation.

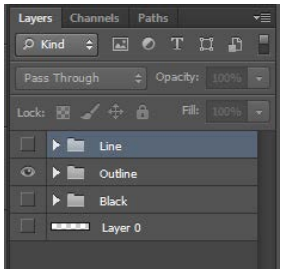
EXPORTING UI FROM PHOTOSHOP

There are 3 types of UI styles you can use. This section is a guide on how to export them if you need to.

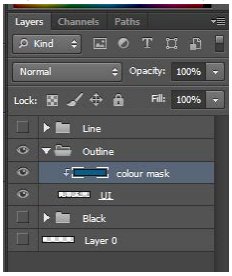
INSTRUCTIONS

1. Open the 'master-ui-spritesheet(colours).psd', which can be found here: \\creativelon1\Creativetest\3 Library\Templates\featureTemplate

2. Choose the style that you want by selecting turning on the appropriate layer group.



3. To change the colour:
    a. Open the folder that you have active and select the colour mask.



    b. Go to the canvas and use the Paint Bucket Tool to fill the layer with your chosen colour.
4. Export the layer as a transparent png called 'spritesheet.png', then paste this image into your features images folder.

UI SWITCHING

Some features might need to switch the UI on certain slides, as they may not be visible on certain backgrounds. To define which slides should be switched, change the `ui_switch` setting under `slider_settings` to either:

a) An integer representing the slide number where the switch is to happen (if only required for one slide)
b) An array with all the slides that require switching

So, for example, if you wanted to perform the switch on slide 2, you would put:

```
ui_switch : 2
```

...and if you wanted to perform the switch on the intro, third and fifth slides, you would put:

```
ui_switch : [0, 3, 5]
```

For this to work out-of-the-box, you will need to create an appropriate spritesheet. Follow these steps to do so:

1. Create a Photoshop canvas that is 232 x 72px (twice the height of the standard sprite). Make sure it has a transparent background.
2. Go through the spritesheet exporting process described in the "Exporting UI from Photoshop" section above for each colour you wish to switch.
3. Drag the resulting pngs into the new canvas that you created, one on top of each other, making sure they don't overlap.
4. Export them as spritesheet.png and put them in your images folder.

CODE PATTERN

The carousel template JavaScript is divided into sections. This section describes what each one does and is laid out in the order that they appear in the code. Understanding this will help you to extend the functionality of the plugin.

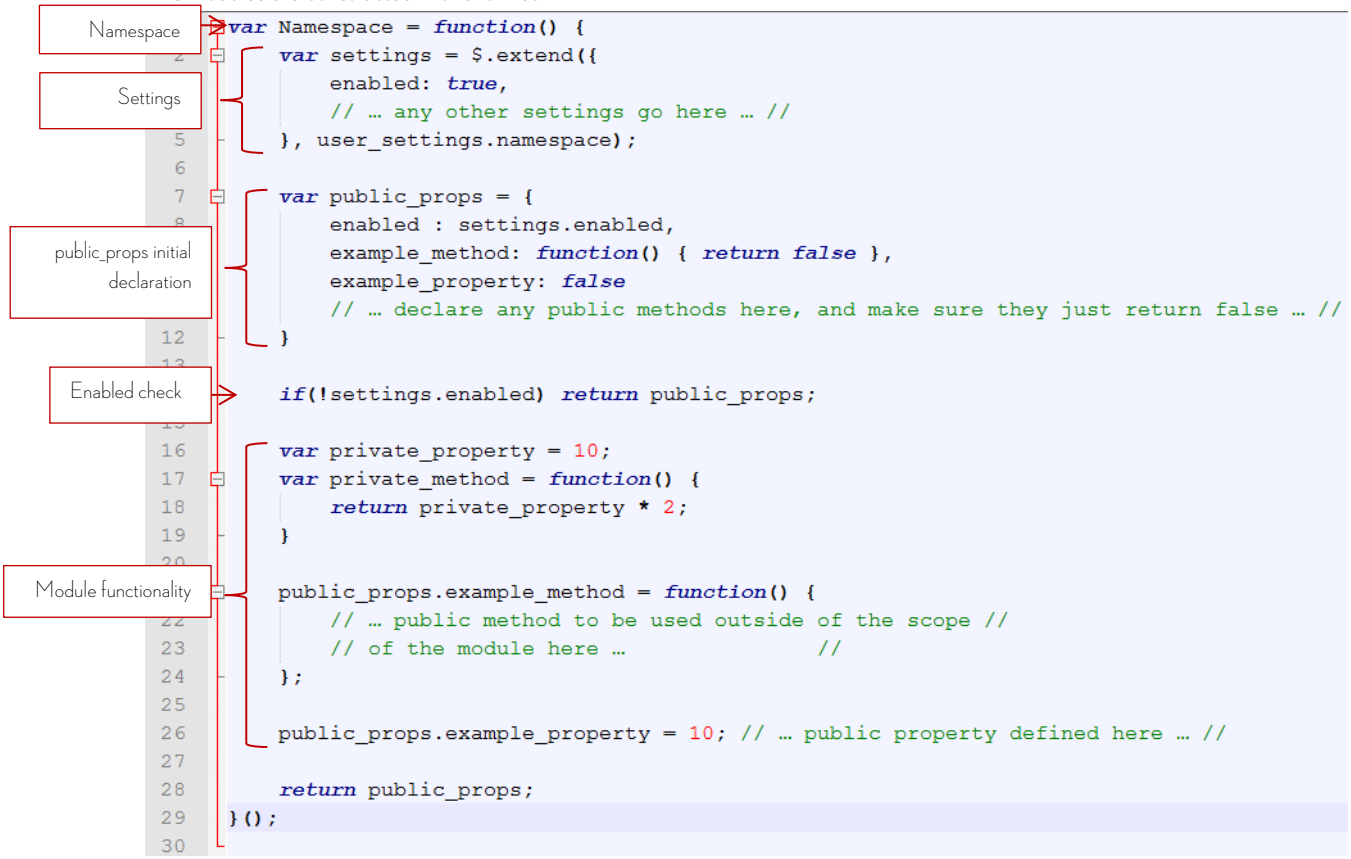
HELPER FUNCTIONS

These functions are functions for common tasks that you can use throughout the plugin. As a general rule, if you need to repeat a generic action more than once, it's worth creating a Helper function here.

MODULES

The plugin uses a modular pattern that is designed to only process the enabled modules, and will skip the disabled ones. As the modules are defined, they should be abstracted from each other, meaning none of the modules should be dependant another to run. They all have access to the helper functions. Their functionality is united in the "Core Processing" section, where they may interact with each other using their public functions and callbacks.

The modules are constructed in this format:




```
var Namespace = function() {
    var settings = $.extend({
        enabled: true,
        // … any other settings go here … //
    }, user_settings.namespace);

    var public_props = {
        enabled : settings.enabled,
        example_method: function() { return false },
        example_property: false
        // … declare any public methods here, and make sure they just return false … //
    }

    if(!settings.enabled) return public_props;

    var private_property = 10;
    var private_method = function() {
        return private_property * 2;
    }

    public_props.example_method = function() {
        // … public method to be used outside of the scope //
        // of the module here …                 //
    };

    public_props.example_property = 10; // … public property defined here … //

    return public_props;
}();
```

Namespace

This is a handle for the module, containing a self-invoking function that returns the public_props object.

Settings

This is the default settings for the module. These are overwritten with any settings that you include when you declare the Carousel plugin.

public_props initial declaration

Here is where you declare the methods and properties that will be available outside the scope of the module. This should at least include `enabled : settings.enabled`, and any other methods or properties you wish to use should be `function() { return false }`. This prevents an 'undefined' error if the module is disabled. The actual methods/properties will be declared later on in the module.

Enabled check

This bit of code checks if the module is NOT enabled, and if so returns the public_props object defined above. If the module is enabled, you can continue to declare the public methods/properties below this point.

Module functionality

After the Enabled check, you can start writing the code for the module's functionality. Private methods and properties can be declared in the normal way, with `var method = function() { … etc.` , while any public methods can be declared by including it in the public_props object. After all the processing is done, the function should return the public_props object.

CORE PROCESSING

This is where the functionality of the carousel is programmed. This is the core of the feature that tells the carousel how to behave. It is based around a bx-slider instance, and uses that plugin's API for the slider functionality and callbacks etc.

If you want to add new functionality to the template, you will need to integrate them with the Bx-slider. The key to this is to call module's methods in the callbacks of the `get_attrs` function in the Core Processing section. This defines the settings for the Bx-slider instance that is created.

HELPER FUNCTIONS

BIND UI

```
77      function bindUI(element, event, callback) {
78          var el = element instanceof jQuery ? element : $(element);
79          el.on(event, callback);
80      };
```

This function is for binding events to UI elements and it takes 3 required parameters:

element

This is the target UI element you wish to bind the event to. This can either be a CSS selector (e.g. '.btn'), or a cached jQuery selector.

event

This sets the event type, this can be any type of event that can be used with jQuery's `.on()` method.

callback

This is the function that fires when the event is detected.

Here is an example of the function in use:

```
308             $toggle_btn = $('#pager-tab'); // Caches toggle button
309
310             // Binds the toggle function to an element
311             bindUI($toggle_btn, 'click', function() {
312                 public_props.toggle_pager();
313             });
```

FEATURE APPEND

```
73      function feature_append(html) {
74          global_settings.feature_wrapper.append(html);
75      }
```

This function appends a string to the `.feature-wrapper` div, and is used to attach UI elements to the feature. It accepts one parameter:

html

This should be a string that will parse as valid HTML when inserted into the DOM. See below for an example of it in use, including the creation of a multi-line JavaScript string by using an array and the `.join()` method.

```
297             var html = [
298                 '<div id="pager-wrapper">',
299                     '<div id="pager-container">',
300                     '</div>',
301                     '<div id="pager-tab"></div>',
302                 '</div>'
303             ].join('\n');
304
305             feature_append(html); // Inserts the HTML
```

TRANSLATION

```
// Determines appropriate language to use
function translation(copy) {
    var mkt = market.slice(0,2);
    var lang = (mkt === 'en') ? 0 : (mkt === 'de') ? 1 : (mkt === 'fr') ? 2 : 0;
    return copy[lang];
}
```

This function takes in an array with the copy for each language inside it, then returns the correct copy for the market. Here is an example of it in use (in conjunction with `feature_append()` helper):

```
var buylook_copy = translation(['Buy complete look', 'Den kompletten Look shoppen', 'Acheter la tenue compl\u00e8te']);
var show_copy = translation(['Show','Anzeigen', 'Montrer' ]);
var hide_copy = translation(['Hide', 'Verbergen', 'Cacher' ]);
var html =
    ['<div id="grid-container">',

        '<div id="grid-ui" style="display: none">',
            '<a href="#" class="quicklink" id="complete-look">' + buylook_copy + '</a>',
            '<a href="#" id="toggle-grid">',
                '<span class="show-hide" id="show-copy">' + show_copy + '</span>',
                '<span class="show-hide" id="hide-copy">' + hide_copy + '</span>',
            '</a>',
            '<br /><img id="grid-loading" src="' + global_settings.img_path +'bx_loader.gif" />',
        '</div>',
        '<ul id="grid">',
            '<!-- Products are dynamically created here -->',
        '</ul>',
    '</div>'].join('\n');
feature_append(html);
```

APPLY QUICKINKS

This function applies the fancybox quick shop to all tags with classes of "quicklink". This doesn't accept any arguments, so you can just call it at any point you want to attach the quickshop functionality.

ADDING MODULES TO THE TEMPLATE

You may wish to add some extra functionality into the template in the future. To do this, follow these steps:

INSTRUCTIONS

1. Create a new namespace handle for the module in the Modules section. The module takes the form of a self-invoking function.

```
// =============== //
// === MODULES === //
// =============== //
var NewModule = function() {

}();
```

2. Create the default settings as an object inside the `$.extend` function so they can be modified. The settings should at least include "enabled" so the module can be switched on and off easily.

```
var NewModule = function() {
    var settings = $.extend({
        enabled: false,
        example_setting_1: 100,
        example_setting_2: 'example'
    });
}();
```

3. Create a handle for the methods and properties you intend to be publicly accessible. At this stage you probably won't know what variables and functions you are going to use, but you can add these to this object once this becomes apparent

when you code the functionality of the module.

```
85      var NewModule = function() {
86          var settings = $.extend({
87              enabled: false,
88              example_setting_1: 123,
89              example_setting_2: 'example'
90          });
91
92          var public_props = {
93              enabled : settings.enabled,
94              example_method : function() { return false },
95              example_property : false
96          };
97      }();
```

4. After the public_props are declared, add the 'Enabled check' code to check whether the module is enabled, and cease all processing if it isn't.
`if(!settings.enabled) return public_props;`
5. After the 'Enabled check', you can now start coding the modules functionality. End the module by returning the `public_props` so they will be accessible outside the scope of the module.

```
85      var NewModule = function() {
86          var settings = $.extend({
87              enabled: false,
88              example_setting_1: 123,
89              example_setting_2: 'example'
90          });
91
92          var public_props = {
93              enabled : settings.enabled,
94              example_method : function() { return false },
95              example_property : false
96          };
97
98          if(!settings.enabled) return public_props;
99
100         // Variables and functions defined here will only be accessible
101         // within the scope of the module
102         var private_var = 200;
103         var private_function = function() {
104             // ... functionality goes here ... //
105         };
106
107
108         // Adding a function or variable to the public_props object will
109         // make them accessible outside the scope of the module
110         public_props.example_method = function() {
111             // ... functionality goes here ... //
112         };
113         public_props.example_property = 234;
114
115         return public_props;
116
117     }();
```

6. To integrate the module with the slide, call the appropriate public method/property in the Core Processing in the necessary place. Here is our example called in the bx-slider callback:

```
// This returns the the attributes for the slider
var get_attrs = function( width, current_index, nav_location ) {
    return {
            slideWidth: width
            ,startSlide: current_index
            ,pager: Pager.enabled
            ,pagerSelector: Pager.container
            ,oneToOneTouch: false
            ,buildPager: function(slideIndex){
                return Pager.generate_thumbnail(slideIndex);
            }
            ,onSliderLoad: function(current_index) {
                NewModule.example_method();
                cmCreateElementTag(global_settings.cm_tag + '_' + current_index, 'OUTFIT');
                settings.initial_slide = current_index;
                set_ghosts();
                Jpeg_animate.playFrames(current_index);
                ProductGrid.update(current_index);
            }
            ,onSlideBefore: function($slideElement, oldIndex, newIndex){
                $('.' + settings.active_class).removeClass(settings.active_class);
                $slideElement.addClass(settings.active_class);
                Pager.toggle_pager(true);
                settings.initial_slide = newIndex;
                Layers.slide($slideElement, viewport_width(), oldIndex, newIndex);
                ProductGrid.update(newIndex, true);
                ui_switch(newIndex);
```

NOTES

- Refer to existing modules to see how they work if you are unsure about anything.
- Some modules might need to add text which will need to be translated automatically – see how this is accomplished in the `ProductGrid` module.
- If you see a function in another module that would be useful in your new module, convert it into a helper function and update each module to use the helper function instead.
- Try to minimise the public properties and only make functions public when they need to be. Making use of initialisation functions can help with this.
- If possible, bind events to elements within the module. There is a `bindUI()` helper function to help with this.