Developer Guide

This Document assumes you are familiar with your Flash IDE of choice in regards to creating new projects and adjusting their compiler/library properties.

Setup

Create a new flash project and add the BMApps.swc file to your project library path. Declare a variable of type BMApplication.

Initiate

As with some other complex web api objects, it must be initiated.

```
//Make an app. Always pass the loaderinfo parameters to the constructor.
//It may contian the hooks the portal uses to transfer players to your game.
brassmonkey=new BMApplication(loaderInfo.parameters);
brassmonkey.addEventListener(DeviceEvent.DEVICE_CONNECTED, onDeviceConnected);
brassmonkey.addEventListener(DeviceEvent.DEVICE_LOADED, onDeviceReady);
brassmonkey.addEventListener(DeviceEvent.DEVICE_DISCONNECTED, onDeviceDisconnected);
brassmonkey.addEventListener(AccelerationEvent.ACCELERATION, onAccel);
//Initiate. Then setup controls.
brassmonkey.initiate("BrassMonkey Flash",4);
```

This gets the object ready to receive control-design data and other post-initiation actions.

But what to do?

Configure the Core and Start

Several things can be done before 'discovery' of your users begins. Brass Monkey is constantly growing and improving its features ans has set up version control for those who target specific features. The first parameter is the major version and the second parameter is the minor version. Refer to our feature matrix for the desired values, however 1.6 is recommended at this time because of the new control pad API's

```
        SettingsManager.VERSION_MINIMUM.minor = 6;

        SettingsManager.VERSION_MINIMUM.major = 1;
```

Most important of all the core settings is what the user sees on the smart-device. What consoles and USB controllers have in common is that they never change from game to game. Developers have created rich navigation and control experiences using only a pair of D-Pads and an 'A/B' buttons. Maybe you develop games for smart-devices already and have your own colorful take on the D-pad. Brass Monkey prefers that developers are given the freedom to design the controls and buttons. We currently do not have any pre-configured control-designs but we do have many examples. Unity provides a WYSIWYG editor for static controls and Flash Professional while not technically an 'editor' allows WYSIWYG with paging/multi views. At run-time, each frame of the movie clip becomes a view in the control-design. In JavaScript, we have not created a visual tool to build them yet but they can be created with simple json. This will be covered in detail later. The design also sets the initial request for touch and acceleration sensor events.

Load a control-design:

```
      // Make our control scheme.
      var controlScheme:ControlScheme = new ControlScheme();
      var appScheme:AppScheme = BMControls.parseDynamicMovieClip(controlScheme);
      //add the contols we made to brass monkey
      brassmonkey.session.registry.validateAndAddControlXML(appScheme.toString());
```

You can load more than one design and create custom designs at any time to personalize the user experience. After the call-back, you would load another if desired. They are stored and referenced by integer index. After you have set up everything and are ready, start 'discovery'.

```
brassmonkey.start();
```

Discovery
Discovery is a process we refer to when we talk about detecting and connecting to smart devices from the browser/console. Connection to the smart-device is initiated by the user. While it did pose a couple challenges to building the system, it allowed for multiple consoles to be present on the same LAN. Your user-experience/game is referred to as the 'Host' and as mentioned before, there can be up to 64 hosts at any single LAN. It may or may not be important to know which id or 'slot' is assigned to your host, but it is very important to the end user since they initiate the game-session using a wireless device.

```
brassmonkey.addEventListener(DeviceEvent.SLOT_DISPLAY_REQUEST, onSlot);
```

This function is called when the host is first assigned the slot, and also when a user taps their smart-device on the host list. The tap from the user signifies that they are trying to identify which screen they are looking at is the one they are going to connect to. In most cases there will only be one host to select from but 'just in case', Brass monkey ensured that the user would be in control of selection.

User life-cycle and when it matters
Just like all 'Web 2.0' applications, the user-reference provided to the host goes through a multitude of 'states'. You probably know that you'll create particular objects to work with users when they join, and destroy them when they depart the host. Or maybe you prefer other mechanisms to grind user references through an experience. There are three events to process. The first event is to set up the state that the device will open with. You could designate game pad, text mode, navigation, or waiting. By default you won't need to use this event, but if you do, this event is here for presetting and getting information about a client before it is fully loaded.

```
brassmonkey.addEventListener(DeviceEvent.DEVICE_AVAILABLE , onDevice);

protected function onDevice(event:DeviceEvent):void
{
      switch(event.type)
      {
            case DeviceEvent.DEVICE_AVAILABLE:
            event.device.controlMode= Device.MODE_GAMEPAD;
            event.device.controlSchemeIndex = 0;
            event.device.attributes['someName']="someValue";
      //IMPORTANT note on using the DEVICE_AVAILABLE event.
      //Hooking this event requires your action to complete connection.
      //This is the only time you are required to manually connect a client.
      //If you do not hook this event, it will connect automatically, provided
      //there is less than the max players already connected.
            brassmonkey.session.connectDevice(event.device);
            break;

      }
}
```

The controlMode property determines which mode, MODE_GAMEPAD, MODE_KEYBOARD, MODE_NAVIGATION, or MODE_WAIT that the client will start with . You can also set an attribute or two to the device for later reference. ' controlSchemeIndex' is only needed if you have loaded more than a single control-pad design.

The next event is generally not used, but signifies that the client was actually reached through the LAN.

```
	brassmonkey.addEventListener(DeviceEvent.DEVICE_CONNECTED, onDevice);
```

The device then requests and begins to load your design using a P2P connection. After the design is processed comes the third event.

```
	brassmonkey.addEventListener(DeviceEvent.DEVICE_LOADED, onDevice);
```

This event signals that the device is ready for scripting and play interaction. If the control-design included 'acceleration' and 'touch' enabled, the data will begin to fly at your host.

The final life-cycle event you will receive normally is the device disconnecting. It is possible that the user closes the browser at any time also which would signal game over.

```
	brassmonkey.addEventListener(DeviceEvent.DEVICE_DISCONNECTED, onDisconnected);
```

Here you clean up objects that were made for the user or remove the users reference from your game engine.

In addition to the event and value properties, in a targeted call-back you will find the referenced device properties. Altogether they amount to:

A note about text input.

Note the difference between a DeviceEvent.TEXT_ENTER and DeviceEvent.KEYBOARD. The CALLBACK_TEXT_ENTER event is when you would process their input. It is triggered by the 'return'/'submit'/'new-line' key or hardware switch. The textContent property of the device object would contain the text as input linearly, not including cursor position. It would be good to display the text on the console screen either as they type it or when it is submitted to ensure that the host gets what the user intended. The typical usage is to put the device in to keyboard mode, and then back into game-pad mode, or previous mode at the DeviceEvent.TEXT_ENTER event. The DeviceEvent.TEXT_ENTER is dispatched whenever the '\n' character code is read by the host from the device while the DeviceEvent.KEYBOARD is dispatched with every key press.

```
	//keys are dispatched from the session.

	brassmonkey.addEventListener(DeviceEvent.KEYBOARD, onKey);

	//'enter' is dispatched by the device directly.

	device.addEventListener(DeviceEvent.TEXT_ENTER, onTextEnter);
```

User Ejection

If you want to remove a device from the host session without user input, you can close it.

```
	device.disconnect()
```

Controls - What do they see in their hand?
The design of the control pad that is sent to the user devices is made up of png/jpg/swf based bitmaps up to ARGB 32-bit color depth. These images are placed on the device screen according to the rectangle provided. The control-design protocol specifies values between zero and one for left/right/height/width. If you had a background jpeg at 24bit depth to cover the entire screen, its values would be (0,0,1,1). The over-all pixel size of the control-design is specified in the root of the design object, or in the header of designs serialized to XML and JSON. The over-all size of the design will be scaled to its maximum size on all devices while maintaining the intended aspect ratio. If your game uses touch events, then the touch-positions are also scaled to the design's intended values. We decided that by allowing multiple control-designs in a session and also scaling designs to the maximum available screen space, the choice to create higher-definition controls for tablets and pads or not, was left to the developer. We also recognized that adjusting screen coordinates values to the design dimensions kept the developer from having to catch and code for arbitrary screen sizes. For instance, in Tank vs Alien, we compute a 24 position d-pad wheel for turret aiming which is based on the touches within a its rectangle. The routine that processes the touch-positions was much simpler, knowing that the values we are looking for will fall in that rectangle.

The control-designs can be made with the Unity IDE, the Flash Professional IDE, by hand in raw text, or even by remote services at run-time. Currently we have three types of components and we are dreaming up others. We have now, BMImage, BMButton, and BMText. We are looking at adding an HTML component that runs in the device web-kit to provide client side scripting support for control-design events. Spinners, rotators, and other fun stuff is on the horizon.

BMImage – v 0.9.0
The base control element is the BMImage. A single static texture that does not respond to tactile input. Mostly used for backgrounds, but also good for logos and accents.

BMButton – v 0.9.0
The BMButton is a BMImage control elements with two states. It uses two skins/textures and responds to user tactile input. When the user presses or releases the BMButton, the host can receive button call-backs with the name of button and the button state, 'up' or 'down'.

BMText – v 1.2.0

BMText allows you to place editable text on the controls. You can write hints or anything your user-story requires. Having editable text on the control-design itself can be a great benefit in many situations. The BMText is a single line, so to use multiple lines you would place several BMText components on the design until the maximum number of text lines desired was reached. See the Dynamic texts demo for more information.

BMHitRects – v 1.4.0

The easiest way to make extended hit rects is to create another graphic that represents the hit-rect over the button. At runtime, remove that particular graphic from your control design and apply its rectangle to the desired button.

```
//we will remove the reference display object from the control pad design

//but then assign their rectangles to the hexagon button.

var bmImageL:BMImage = appScheme.removeChildByName("leftRect") as BMImage;

//Grab the hexagon button references by name.
var buttonL:BMButton= appScheme.getChildByName("left") as BMButton;
//apply the different hit rect to the button.
buttonL.hitRect=bmImageL.rect;
//validate the controls.
brassmonkey.session.registry.validateAndAddControlXML(appScheme.toString());
```

BMControlMenu – v 1.6.0

When the client slides the settings icon to activate the control-menu, it is possible to put in your own settings menues and update them dynamically.

See the ControlMenu demo for more information. It loads an online radio station in to the browser, and allows you to mute/unmute the volume.

Changing the layout at run-time

Starting with application version 1.2, you can send changes to the layout of the control-design. The underlying protocol allows you to hide or move items but it does not provide for any other type of view organization or control over individual elements. The Flash and JavaScript endpoints have extended the specification by adding a 'page' and 'name' property to control elements. The control-design views are paged from the host to the devices by this property. While you can manage design views at run-time in many ways on your own, it takes an advanced familiarity with the internal protocol. The Flash and JS libs give you an easy way to organize your designs and how they are presented.

Layout

By default, the devices will start on page 1 of your control-design. To flip to page 2 call:

```
brassmonkey.session.updateControlScheme(device,appScheme.pageToString(2));
```

The device will be updated to show only the control-design elements marked page 2.

Text

BMText is changed at run-time by referencing the element name, the new text, and also signal to change 'now' or not. The Boolean signal to update the device 'now' is present so that you can either, progressively or all at once, update several text elements.

```
var bmText:BMDynamicText=appScheme.getChildByName("dynamicText") as BMDynamicText;

bmText.text = "New Text ";

brassmonkey.session.updateControlScheme(device ,appScheme.pageToString(1));
```

Make sure the text element referenced is on the current displayed page. If you use the Flash Professional IDE to create designs, make sure to name text elements uniquely on every frame. The IDE will not prevent you from placing text fields on different frames using the same instance name. Each TextField in each frame/page will need a unique name to support this paging system. Remember though, that nothing prevents you from developing additional ways to update the control-design layout independent of the page method.

Best Practices and Advanced Concepts using dynamic control-designs

The number one 'best practice' is to send all resource to the client-device up front. You can generate new resources at run-time during the client session and send them over, but the speed at which devices process the resources is unpredictable. We hope that with dynamic text and layouts, the cases where resources are generated late will be significantly less.

Brass Monkey does encourage generating resources at run-time during discovery. When a device requests a connection to your host, you will receive the deviceId and friendly name. You could whip up a background image that incorporated their device's name and send it in the initial control-design. Using the method below, the resources will need to be loaded and encoded prior to process, but the end result is incrementing the controlScheme count and the new device requesting the session can have this new index specified as its unique personalized design.

Identifying your Users

Identifying users in a web game usually involves tying into the HTML session for log-in status, or looking at the present document context. While that is all still usable, that the web game can join with multiple players adds the need for a reliable reference. The device object presents the 'deviceId' property as our GUID reference. The transient attributes also help you keep track of your player states. The first step in getting your game access to a more robust tool set is to register your application with Brass Monkey. Contact developer.services@playbrassmonkey.com to receive your application GUID. With the GUID you can write cookies onto individual devices and read them back in future sessions.

```
brassmonkey.session.setCookie(event.device, "somePropName","somePropVal");
```

To read it back, call:

```
//add a listener to the device.
device.addEventListener(DeviceEvent.GOT_COOKIE, onCookie);
brassmonkey.session.getCookie(event.device, "somePropName");
```

When cookies are read from the remote device, the values are copied to the local transient attributes property of the device object reference.

```
protected function onCookie(event:DeviceEvent):void
{
	//All cookie values retrieved will be cached in
	//the device attributes property.
	//trace them out
	for(var prop:String in event.device.attributes)
	{
		trace("attribute -",prop,":", event.device.attributes[prop]);
	}
}
```

Brass Monkey lets users register one or more smart-devices to the same user profile. If you want to store attributes for a user that are readable with every smart-device they have registered, you can use the Host-Session API.

The advanced usage is not covered here yet, but the basic usage is the following.

```
//this call does not require a secret key/handshake.
HostSession.getIconByDevice(event.device, onIcon);
```