**DwBP - 3 - Using JavaScript with Brightcove Player**

There are a few JavaScript concepts I want to be sure you understand before we jump into development. These are key tools from JavaScript that are used frequently in Brightcove Player development. Please know that if you have an understanding of callback functions and event driven development framework, you really do not need to watch this video.

Before we actually look at the JavaScript concepts, I need to mention an issue about how to implement the code we will write. Initially for the development courses of Brightcove, we followed many best practices and design patterns. What we found is that for some attendees, we obscured the Brightcove development with JavaScript issues.

For this reason, the code we show stresses the Brightcove API. You may wish to implement the code in your favorite manner.

This slide shows a few bullets about the module pattern, a favor pattern of ours to use in more complex JavaScript code.

The Brightcove Player API is an event driven framework. As you can see by the definition, this means that the program behavior is driven by the production, detection, and consumption of events. In simpler terms, everything that happens with Brightcove Player takes place because an event is dispatched, handled, and associated event handler code is then executed.

Program execution behavior is not as you see it on the bottom left. Where line 1, line 2, line 3, et cetera are executed in order.

Actual program execution is shown on the bottom right. Where an event, like when the player is ready or the video starts playing, is dispatched. And then an event handler function is called and code is executed.

Another thing that we have to understand are callback functions. This is standard JavaScript technique and not something particular to Brightcove. But it's a really a good thing we have this callback functionality in JavaScript the Brightcove engineers can use. And here's why.

Consider the method called getVideo(). I could ask the question, how long is it going to take to get requested video back to your computer device after the method is called? This can be a confusing question because there is not a single answer. The answer is, of course, it depends. And that is exactly why we have to have a callback function.

So what happens when I do call getVideo()? The method sends a request to a Brightcove server for a video object. At some point, the data will be returned. Only at this point will the callback function be called.

So no matter how long it takes, a callback function is not called until the video object gets returned to the client.

By the way, there is a timeout of 45 seconds before an error is displayed if the video is not returned.

I don't want to get too much into the details, but callback functions need to be thoroughly understood, as you will use them often when developing with Brightcove player. We need to decide how to write a callback functions. There are three options. As an anonymous function, as a function declaration, and as a function expression.

The anonymous function is a common technique to use for callback. This keeps a call to the function and its associated callback function together. This is not a good technique when you want to reuse the same callback function for multiple other method calls, as it is not reusable when written as an anonymous function. The previous slide showed an example of an anonymous function.

The function declaration is what most people would think of as the normal way to define a function. An example will be shown shortly. The good thing is that the function will be loaded before any code is actually executed. The bad thing is this technique could lead to JavaScript collisions and is it possible to have name conflicts if not implemented carefully. The module design pattern mentioned earlier is a good way to avoid name conflicts.

The function expression defines a function as a variable. An example will be shown shortly. If you try to call a function expression before it's loaded, you'll get an error because function expressions load only when the interpreter reaches that line of code. The good thing is you can scope function expressions as you do any other variable.

You see in this example an anonymous function. The callback function is defined as the argument of the getVideo() method. And the function has no name.

Here you see a function declaration. Note that the function can be called before it is defined as JavaScript will look for all function declarations before executing any code. The function is defined separately from the call to getVideo(), so it can be reused.

Lastly is a function expression. Note that the function is defined before it is used. The function is defined separately from the call to the getVideo() method, so it could be re-used.

Now we've built a good foundation. So let's get coding.