Chapter 9: Asynchronous Operations

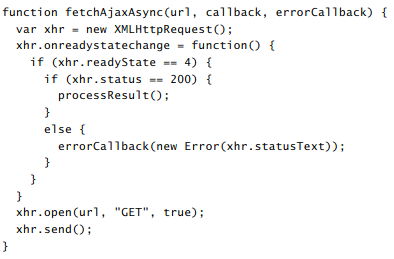
Contents

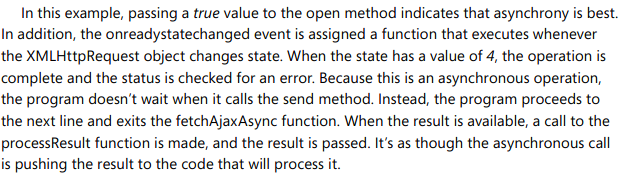
**No table of contents entries found.**

# Let’s write some Asynchronous Operations!!!!!

## Let’s go back to Zapp’s class and learn how AJAX really works!!!

The example below is awful and we shouldn’t have to learn it. My hope is that we don’t have to actually know this and it’s just there. In case you actually want to know what it’s about, you can read the code example below.





## We need a promise object!

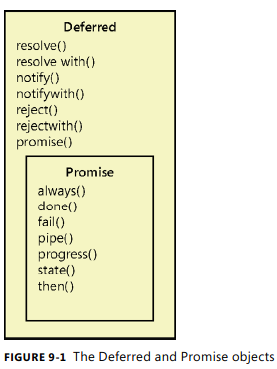
The promise object can be in one of three states:

1. Pending – Starting
2. Resolved or Rejected – Move from starting to here

**jQuery has promise objects!!!**

The method that jQuery uses for creating a promise is called $.Deffered(). The deferred object contains a read-only promise object.

The creator of the promise objects controls which state (pending, resolved or rejected) that it is in by calling a method on the $.Dereffed() object. In addition to this, the following methods are available.



## Workflow of an Asynchronous operation in JQuery

1. Create your asynchronous method using the deferred object. The example below sets a timeout using a deferred object. The code deferred.resolve() is executed when the timeout expires.

function timeoutAsync(milliseconds) {

var deferred = $.Deferred();

setTimeout(function () { deferred.resolve(); }, milliseconds);

return deferred.promise();

}

1. Subscribe to the completion of timeoutAsync using the returned promise object.

function subscribeAsync() {

var promise = timeoutAsync(2000);

promise.done(function () {alert (‘done!’) });

return promise;

}

1. Call subscribeAsync which calls timeoutAsync with a value of 2000. We get the promise from timeoutAsync, so can subscribe to when the deferred object resolves (the timer expires) that we call alert(‘done!’).

An example of how this works can be found here: <https://jsfiddle.net/wwxd96n4/3/>

## You will fail! Know how to fix it!

You can subscribe to the failure of your asynchronous call. It’s really easy to do!!! Simply call the following!

promise.fail (function () {

// fail gracefully here }

);

## Clean up your Async calls!!!!

It’s good to keep things clean. Daily baths, sweeping the floor.. So why wouldn’t you clean up your asynchronous calls!?!?!? You would, so I will show you how!

promise.always( function () {

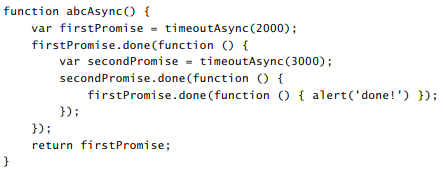
// write code that will always execute after the call fails or resolves!

} );

## Chaining async calls – because it’s fun!

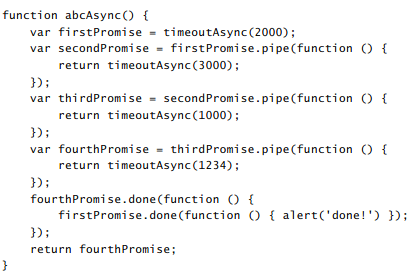
So, you want to run two nested asynchronous calls such that one depends on the other. Or maybe you want to run three or one-thousand. Well, I’m here to tell you that’s not possible. Actually I lied, it’s totally possible.

You could nest your asynchronous calls like the image below. Maybe this will work for two or three, but it quickly gets out of hand. Also, what happens when the firstPromise finishes before the secondPromise? Not good things, that’s for sure.

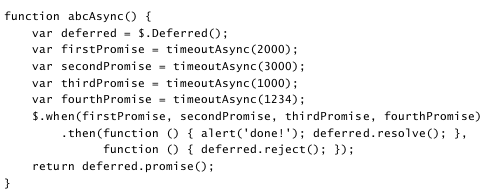


Fortunately, smart people have solved this problem!! **We can use the Pipe Method!!!!!!** The pipe method waits until the previous operations is completed before starting the next. An example of how to do this is found below.

<https://jsfiddle.net/jwjzj5Le/3/>



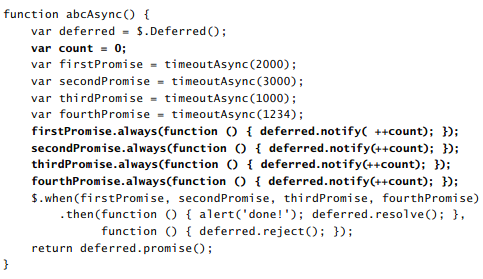
The example above shows blocking calls where we need to finish executing the firstPromise before we can begin executing the secondPromise and so forth. We can remove the blocking and do everything in parallel using $.when().then() as shown in the example below.

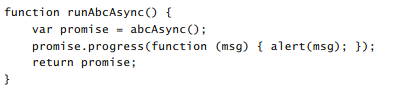


This works because the $.when method returns a resolved promise object.

## Displaying progress of an async call!!!

You can display the progress of an async call!! Just call the notify method on the deffered object!! The promise object can subscribe to the notify method by calling the progress method! The example below shows how this works.





# Web Workers are great for SPA apps!!!!

Web Workers are JavaScript files that can perform operations in the background by pushing data to your main JavaScript class. Basically the publisher subscribe pattern, where the worker subscribes to events from your main class and pushes data back when it is done. The worker is always waiting for you to send it something to do.

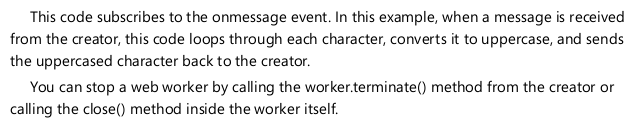
## Workers are live in a cave, very isolated from the world

They don’t have much.. in fact, they don’t have access to any of these things..

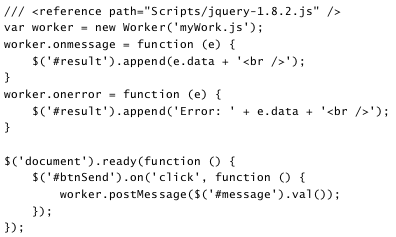
* Mutexes
* Semaphores
* Critical Sections
* Synchronization Locks
* DOM

Basically all the cool things we learned about in OS.. They don’t get to use any of them! So they are kind of unsafe to use.. depending on what you are doing!

## Let’s look at an example!



Main.js



Worker.js

