Most of the time server-side programs sends back a single piece of data, like -1 or “okay.”

But sometime server needs to send a MULTI-valued response at one time, lile:

* A string description of the selected item.
* A numeric price for the item, like 299.95.
* A list of URLs with related information about the item.

Theses all together in single response.

How is this possible

* XML
* CSV
* innerHTML:

Suppose you had the following information for an item:

* Item ID: itemCowbell
* Description: Remember the famous “more cowbell” skit from Saturday Night Live? Price: 299.99
* URLs: http://www.nbc.com/Saturday\_Night\_Live/
  + http://en.wikipedia.org/wiki/More\_cowbell

How would a server represent this information?

...as XML

<?xml version="1.0"?>

<item id="itemCowbell">

<description>Remember the famous "more cowbell" skit from Saturday Night

Live? Well this is the actual cowbell.</description>

<price>299.99</price>

<resources>

<url>http://www.nbc.com/Saturday\_Night\_Live/</url>

<url>http://en.wikipedia.org/wiki/More\_cowbell</url>

</resources>

</item>

* We used an attribute for the item ID.
* The description and price are in XML elements
* We grouped the URLs with a resources element, and then put each URL in a url element.

as CSV(comma separated values)

itemCowbell,Remember the famous 'more cowbell' skit from Saturday Night Live? Well this is the actual cowbell., 299.99, <http://www.nbc.com/Saturday_Night_Live/>, http://en.wikipedia.org/wiki/More\_cowbell

Each item in a CSV string is separated by a comma.

3 ...as an XHTML fragment?

<p>Description: Remember the famous 'more cowbell" skit from

Saturday Night Live? Well this is the actual cowbell.</p>

<p>Price: $299.99</p>

<ul>

<li><a href="http://www.nbc.com/Saturday\_Night\_Live/">

http://www.nbc.com/Saturday\_Night\_Live/</a></li>

<li><a href="http://en.wikipedia.org/wiki/More\_cowbell">

http://en.wikipedia.org/wiki/More\_cowbell</a></li>

</ul>

This is the XHTML exactly as it needs to be inserted into the rock and roll page.

CSS styles it, and the data from the server is wrapped up in XHTML tags.

innerHTML is only simple for the CLIENT side of a web app

From a client-side point of view, innerHTML is pretty simple to use.

You just get an XHTML response from the server, and drop it into a web page with an element’s innerHTML property.

detailDiv = document.getElementById("description");

detailDiv.innerHTML = request.responseText;

innerHTML is not a good choice as server response

The problem is that the server has to do a lot of extra work. Not only does the server have to get the right information for your app’s request, it has to format that response in a way that’s specific to your application.

In fact, that format is specific to one individual page on your site!

server-side programmers don’t really like to mess around with XHTML. That’s the

whole reason a lot of these folks move over to the server-side in the first place... no XHTML.

You’re not going to be able to get server-side guys to write XHTML, and then change it all the time on top of that.

XHTML is a mess to work with on the server, and it changes all the time.

What is CSV again?

A: CSV stands for comma-separated values. It just means that several values are put together into a single string, with commas separating each individual value.

In fact, you can use anything you want to separate the values: a pipe symbol (|), an asterisk (\*), or anything else that’s a fairly uncommon character, and change your client code to split on that new character instead of commas.

In fact, CSV is a bit dangerous because an item description might have a comma in it. In that case, you’d end up splitting the description on the comma, and having all sorts of problems.

Q: What is setAttribute()? I’ve never seen that before.

A:setAttribute() creates a new attribute on an element. The method takes two arguments: the name of the attribute and its value. If there’s no attribute with the supplied name, a new attribute is

created. If there’s already an attribute with the supplied name, that attribute’s value

is replaced with the one you supplied to setAttribute().

Q: What about childNodes? What’s that?

A:childNodes is a property on every DOM node. The property returns an array of all the child nodes for that node.

So you can get an element’s children, for example, and iterate over them or delete them.

Q: So why did you iterate backwards over the childNodes array?

A: That’s a tricky one. Here’s a hint to get you thinking in the right direction: when you

call removeChild(), the node you supply to that method is removed from its parent immediately.

That also means that all references to that now-removed node—say in an array full

of an element’s child nodes—have to be updated. Without a child to point to, all the

child nodes that come after the removed node have to be moved up in the array.

So if you iterated over an array like childNodes from front to back, removing nodes as you went, what would

function displayDetails() {

detailDiv = document.getElementById("description");

var response = request.responseText;

var itemDetails = response.split(",");

//This creates a new <p> with the description of the item in it.

var descriptionP = document.createElement("p");

descriptionP.appendChild(document.createTextNode("Description: " + itemDetails[1]));

detailDiv.appendChild(descriptionP);

//Adding another <p> with the price..

var priceP = document.createElement("p");

priceP.appendChild(document.createTextNode("Price: $" + itemDetails[2]));

detailDiv.appendChild(priceP);

//display the URLs as list items in an unordered list.

var list = document.createElement("ul");

for (var i=3; i<itemDetails.length; i++) {

var li = document.createElement("li");

//Each URL goes into an <a>, which is added as the content of an <li>...

var a = document.createElement("a");

a.setAttribute("href", itemDetails[i]);

a.appendChild(document.createTextNode(itemDetails[i]));

li.appendChild(a);

//the <li> gets added to a <ul>...

list.appendChild(li);

}

detailDiv.appendChild(list);

}

“XHTML is just a flavor of XML”.

A flavor of XML is like a specific implementation of XML, with certain elements and attributes defined. So XHTML uses elements like html and p and div, and then those elements are used along with attributes and text values. You can’t make up new elements, but instead you just use the ones already defined.

With XML, you can define flavors like this—sometimes called XML vocabularies—and extend XML for whatever your needs are. That’s why XML is so flexible: it can change to match the data it represents.

If you respond in XML, LOTS of different applications can work with that XML response.

You can use the DOM to work with XML, just like HTML. In fact, the DOM is really designed to work with XML from the ground up.

Even better, the request object you’ve been using to talk to the server has a property that returns a DOM tree version of the server’s response. That property is called responseXML.

var responseDoc = request.responseXML;

function displayDetails() {

var detailDiv = document.getElementById("description");

// Remove existing item details (if any)

for (var i=detailDiv.childNodes.length; i>0; i--) {

detailDiv.removeChild(detailDiv.childNodes[i-1]);

}

// Add new item details

var responseDoc = request.responseXML;

var description = responseDoc.getElementsByTagName("description")[0];

var descriptionText = description.firstChild.nodeValue;

var descriptionP = document.createElement("p");

descriptionP.appendChild(

document.createTextNode("Description: " + descriptionText));

detailDiv.appendChild(descriptionP);

var price = responseDoc.getElementsByTagName("price")[0];

var priceText = price.firstChild.nodeValue;

var priceP = document.createElement("p");

priceP.appendChild(

document.createTextNode("Price: $" + priceText));

detailDiv.appendChild(priceP);

var list = document.createElement("ul");

var urlElements = responseDoc.getElementsByTagName("url");

for (var i=0; i<urlElements.length; i++) {

var url = urlElements[i].firstChild.nodeValue;

var li = document.createElement("li");

var a = document.createElement("a");

a.setAttribute("href", url);

a.appendChild(document.createTextNode(url));

li.appendChild(a);

list.appendChild(li);

}

detailDiv.appendChild(list);

}



Very important points:

* Just as the browser sees your HTML as a DOM tree, web browsers automatically convert any XML they have to deal with into DOM trees too.
* You can work with more than one DOM tree in the same JavaScript function.
  + For example, you can read an XML DOM tree and update an HTML DOM tree, all at the same time.
* HTML elements and XML elements are both just element nodes in the DOM. There’s no difference between an XML type and an HTML type, At least when it comes to the DOM.
* The responseXML property always returns a DOM document object, even if that object is a single element, or just a single text node.

Change Request:

If you ask the server for details about a guitar, you’ll get a manufacturer and year. Clothing?

A manufacturer, sure, but also a size. And for bands, you’ll get a band name, and possibly

the name of the individual in the band that the item belonged to or is associated with.

Solution in CSV :

You need to rewrite the callback function.

assume that every other value is a category, like “Description” or “Price.”

And the values after each category are the actual category values, like the textual description, or 399.99,

or whatever.

not too bad, Except for cases where there’s more than one value, like for those URLs.

Then check(first put) for a special character before each category to indicate that it’s a multi-value category.

CSV is Inflexible

You can’t always know in advance what the data structure you get from the server will look like.

And even if you do, that format might change at anytime.

This is not easy to handle with CSV.

Solution in XML :

Data to be present on website.

* Item ID: itemGuitar
* Manufacturer: Gibson
* Model: Les Paul Standard
* Description: Pete Townshend once played this guitar while his own axe.
* Price: 5695.99
* URLs: http://www.thewho.com/
  + http://en.wikipedia.org/wiki/Pete\_Townshend

Corresponding XML representation:

<?xml version="1.0"?>

<item id="itemGuitar">

<category>

<name>Manufacturer</name>

<value>Gibson</value>

</category>

<category>

<name>Model</name>

<value>Les Paul Standard</value>

</category>

<category>

<name>Description</name>

<value>Pete Townshend once played this guitar while his own axe

was in the shop having bits of drumkit removed from it.</value>

</category>

<category>

<name>Price</name>

<value>5695.99</value>

</category>

<category type="list">

<name>URLs</name>

<value>http://www.thewho.com/</value>

<value>http://en.wikipedia.org/wiki/Pete\_Townshend</value>

</category>

</item>

Close Observation:

* <item> is the root element. It’s the container for all the <category> elements, just like the <html> element in an XHTML file.
* Every category has a <name> and a <value>. They contain the actual data we need to display display.
  + The XML can contain as many <category> elements as necessary. We don’t need to know how many there are or what they are in advance.
* We can have multi-valued categories(URL), and even indicate that with an attribute on the <category> element.
  + Only URL is a category of type(as attribute) list all other category has no attribute.

Algo:

From response get all the category elements.

Iterate all the category elements and Check the attribute of category is list or not.

If it is null then simply extract the name value pair and create p element and add it to detailDiv.

If it is list that means it is URL and has multiple values, so extract the name and associated values and create p element and add it to detailDiv.

function displayDetails() {

var detailDiv = document.getElementById("description");

var responseDoc = request.responseXML;

//get the entire categorie

var categories = responseDoc.getElementsByTagName("category");

for (var i=0; i<categories.length; i++) {

var category = categories[i];

//get the name and type of each category.

var nameElement = category.getElementsByTagName("name")[0];

var categoryName = nameElement.firstChild.nodeValue;

var categoryType = category.getAttribute("type");

//check the type to see if it’s a list.

if ((categoryType == null) || (categoryType != "list")) {

// If not list, get the value, create a <p>, and add text with the category name and value.

var valueElement = category.getElementsByTagName("value")[0];

var categoryValue = valueElement.firstChild.nodeValue;

var p = document.createElement("p");

var text = document.createTextNode(categoryName + ": " + categoryValue);

p.appendChild(text);

detailDiv.appendChild(p);

} else {

// Handling lists of values.i.e. URL.

First, get all the values; For each value, add an <li> to an unordered list (<ul>).

var p = document.createElement("p");

p.appendChild(document.createTextNode(categoryName));

var list = document.createElement("ul");

var values = category.getElementsByTagName("value");

for (var j=0; j<values.length; j++) {

var li = document.createElement("li");

li.appendChild(

document.createTextNode(values[j].firstChild.nodeValue));

list.appendChild(li);

}

// add both the list heading and the list itself to the <div>.

detailDiv.appendChild(p);

detailDiv.appendChild(list);

}

}

}

Q: So the big deal about XML is that it describes itself? That can’t be useful all that often...

A: Actually, self-describing data is useful in a number of situations.

It’s pretty convenient to be able to define elements and structure that’s suited to your business.

Even better, XML is a standard, so tons of people know how to work with it.

That means your vocabulary is usable by lots of programmers, in client-side and server-side programs.

Q: Wouldn’t it be easier to just make up our own data format?

A: It might seem that way at first, but proprietary data formats—ones that you

make up for your own use—can really cause a lot of problems.

If you don’t document them, people may forget how they work.

And if anything changes, you need to make sure everything is up-to-date:

the client, the server, the database, the documentation… that can be a real headache.

Q: Doesn’t XML become a “proprietary data format” when we start declaring element names?

A: No, not at all. That’s the beauty of XML: it’s flexible.

The server and the client need to be looking for the same element names,

but you can often work that out at run-time.

That’s what’s meant by self describing:

XML describes itself with its element names and structure.

Two DOM trees to work with, and dealing with white space in the server’s response?

**JSON** - JavaScript Standard Object Notation

JSON is a way to represent a JavaScript **object** in plain **text**. So the server can send us JSON—which is just text, no XML or DOM issues to deal with—and our JavaScript can work with that response as an object.

You can send and receive JSON from other languages like PHP, C#, Python or Ruby.

The CSV data is pure text. The server sent over the text, and our JavaScript had to use string manipulation routines like, split(), to turn the string into individual pieces of data

itemDetails = response.split(",");

With XML, the server sent text over, too, but that text was self describing.

So we could get a DOM representation of the text using the request object’s responseXML property.

But then we had to use all those DOM methods to work with the object, instead of actual property names like description or urls.

responseDoc = request.responseXML;

With JSON we get text from the server, and then treat that text as a JavaScript object.

Instead of using string manipulation or DOM methods, we’d just use code like item.description or itemDetails.urls.

In other words, we’d have a format that was represented as text for easy network transmission, but an object when we needed to work with the data.

description = item.description;

When you get JSON data from a server or some other source, you’re getting text, but that text can easily be turned into a JavaScript object. Then, you can access that object’s fields using dot notation.

When a server sends its response as JSON data, that data comes across the network as text.

So you can get that data using the responseText property of your request object:

var jsonData = request.responseText;

JSON String:

itemDetails = {

"id" : "itemShades",

"description" : "Yoko Ono's sunglasses. ...",

"price" : 258.99,

"urls" : ["http://www.beatles.com/",

"http://www.johnlennon.com/",

"http://www.yoko-ono.com/"]

}

JSON-String explained:

* The curly braces, { and }, define an object, which is an unordered set of name/value pairs.
  + Each name/value gets turned into a property of a new object.
* Square brackets, [ and ], indicate an ordered array.
  + The urls array gets turned into a property with an array holding its values.
* In your code, you reference elements inside curly braces by their name, but the ones inside square brackets are referenced by number.
  + itemDetails.price is 258.99
  + itemDetails.urls[1] is 258.99 "http://www.johnlennon.com/

Converting JSON string into JAVA-Script Object:

When you run eval()[a Java-script method] on text that describes a set of property names and values, then JavaScript returns an object representation of those properties and values.

There’s just one thing to watch out for. You need to make sure that the overall JSON response string is seen as a single object. So when you call eval(), wrap the whole response in parentheses.

var itemDetails = eval( '(' + request.responseText + ')' );

function displayDetails() {

var detailDiv = document.getElementById("description");

var itemDetails = eval('(' + request.responseText + ')');

for (var property in itemDetails) {

var propertyValue = itemDetails[property];

if (!isArray(propertyValue)) {

var p = document.createElement("p");

p.appendChild(

document.createTextNode(property + ": " + propertyValue));

detailDiv.appendChild(p);

} else {

var p = document.createElement("p");

p.appendChild(document.createTextNode(property + ":"));

var list = document.createElement("ul");

for (var i=0; i<propertyValue.length; i++) {

var li = document.createElement("li");

li.appendChild(document.createTextNode(propertyValue[i]));

list.appendChild(li);

}

detailDiv.appendChild(p);

detailDiv.appendChild(list);

}

}

How to handle dynamic change-request:

In compiled languages, you define your objects in a source file, like a .java file. Then, you compile those files into byte-code. So once your program’s running, you’re stuck with the definitions that are compiled into bytecode.

JavaScript, isn’t compiled; it’s interpreted. Things can change at any time. Not only that, but the objects the server sends are created at runtime, using eval(). So whatever’s sent to our JavaScript, that’s what we get in the itemDetails object.

We don’t need to change our object at all! We just need to know how to figure out what’s IN the object.

You can access an object’s members... and then get an object’s values with those members

How to inspect (for properties) Javasrcipt Object or JSON-String

Suppose you’ve got an object called itemDetails, and you want to know what properties itemDetails has. You’d use this code to get those properties:

for (var property in itemDetails, {

alert("Found a property named: " + property);

}

So the variable property would have values like id, description, price, and urls.

But we don’t want just the property names; we also want the values for each property.

JavaScript lets you access an object’s properties as if the object were an array. But instead of supplying

an array index, like itemDetails[0], you supply a property name, like itemDetails["price"].

In other words, the value returned for itemDetails["price"] for an object is the value of the property named price in that object.

itemDetails["urls"][0]

The value of itemDetails[“urls”] is an array...

way to see if a value is an array.

What about itemDetails[propertyName]? Is the value for that property an array or a single value, like a string?

Unfortunately, JavaScript doesn’t give you a simple way to check and see if a value is an array.

To help you out, here’s a little Ready Bake Code that will tell you if a value is an array or not.

function isArray(arg) {

if (typeof arg == 'object') {

var criteria = arg.constructor.toString().match(/array/i);

return (criteria != null);

}

return false;

}

Arrays are all considered objects by JavaScript.

All arrays have a constructor with the word “array” in that constructor.

Any object that has that constructor is an array.

eval() is not safe to use, It’s not a good idea to run a code that someone else gave you.

* it can be malicious code, like a script that hacks the user’s browser or something. Or
* it can redirects their browser to a porn site.
* It can’t be a correct JSON always. What if there’s an error?
* Evaluating code with an error in it is going to generate errors for the users.

eval() evaluates what you give it, WITHOUT regard for the results of that evaluation.

eval() is an important part of JavaScript.

If you need to pass textual data to another function for evaluation, or even between scripts, eval() is really helpful.

However, eval() can be a problem when you’re evaluating data that you can’t control, like from someone else’s program or server.

You need to PARSE the server’s response, not just EVALUATE it

JSON parser

JSON website at http://www.json.org provides a JSON parser that does all of these things, and more.

You can download a script from json.org called json2.js, and then use this command to parse JSON-formatted data:

var itemDetails = JSON.parse(request.responseText);

This JSON object is created when json2.js(javascript file) is first loaded by the web browser.

parse() takes in a string(server’s response:request.responseText ) and returns an object if the string is valid JSON-formatted data.

json2.js is free and open source. You can actually read through the source code at http://www.json.org/json2.js, and see what it does for yourself.

Web-Security:

Security is ALWAYS a concern when you’re programming for the web.

Always thoroughly test any code that you don’t have complete control over.

A JSON parser keeps your code safer than eval(), but that doesn’t mean you can completely relax. When you’re writing web code, security is always an issue.

In the case of JSON data, JSON.parse() will ensure you’ve got valid JSON data, but you still don’t know what that data actually is.

So you may still need additional checks before using the data in the rest of your scripts.

Anytime you use code from another source, like json2.js from http://www.json.org, you should thoroughly test out the code.