Welcome to module 3, Using SuiteScript Objects. In this module, we're going to look at what the context object is and how we can use that so that we can extract information out of our record. So what we're going to cover in this model is that we're going to look at how we can add custom fields **in** NetSuite. So this is not exactly SuiteScript, but it is part of the customization that we will be building. So creating custom fields is actually part of the SuiteBuilder platform, so we will get introduced to that.

We're also going to look at the context object. So we kind of saw that when we tried to build our Hello World script, except we didn't really use that context object. So now, we're going to look at the information that we can pick up from that context object.

Also, we're going to extract values off of a record object. So that record object will come from that context object, and we're going to see how we can use those record objects to manipulate our form. And then we're also going to look at how we can debug SuiteScript. Now, if you are going to debug a script, then you would have to use a special tool that is specifically for debugging SuiteScript.

The context object-- so what is that? So the context object basically is an object that is passed from the server to our function, so we can get additional information about that particular function. So for example here, we have a beforeLoad script, so in that beforeLoad script, you can see that function has this object, context.

Now, if you are using the template in our SuiteCloud IDE, you might also see that or refer to that as the script context. The documentation sometimes also refers to that as script context, but personally, I prefer to shorten that a little bit. So instead of using script context, I just change that to context so that it's easier to write. But basically, what that context object is, is that it's just a way for the system to give us the information about the entry point.

Now, it is called the context object because it will change depending on the context of where your function is executed. For example, beforeLoad is executed whenever you load a record, so you will be given information about when a record is loaded-- for example, a reference in the form, maybe a reference to the record, things like that. If you save the record, you would be given a different set of information.

Some information are similar, especially if you are using the same script type, but some information are different. So if you're using, say, an afterSubmit or beforeSubmit or whatever entry point, again, don't worry about those entry points for now, but we're going to talk about that in the next module. But basically, just look at the context object as where you're going to get information based on this particular execution.

Now, in SuiteScript 1.0, if you have seen that previously, instead of using the context object, you would have a series of parameters. On a beforeLoad function, I believe we have one parameter. On an afterSubmit function, we have three parameters, I think, so things like that. So instead of having multiple parameters in your function, you would just be given one context object, and all of those parameters or values are inside that context object. So that's something that has changed from SuiteScript 1.0 to SuiteScript 2.0.

Now, regarding the context object, so once you have gotten the context object, you can get the record object from context objects. So for example, if you are running your script on a sales order record, you would get all of the fields on your sales order record. So for example, you have the standard fields, and when we say standard fields, we're referring to fields that has been created by NetSuite and is pretty much standard throughout all accounts. So regardless of the company that's using NetSuite, they would have that field, so that's considered as a standard field.

Now, sometimes, if you have a standard field, there are some information that doesn't really fit that standard field. For example, what if your company uses a coupon code that is tied to a specific sales order? Now, since we don't have that in the system, that's something that you can actually build, so those are called custom fields.

Now, a custom field can be added on any kind of record type. It can be added to an entity record or a transaction record, in this case, or an item record. It just allows more flexibility to people who would be customizing the account because we know that even if two businesses are the same or similar, there are some things that would be unique to each particular business. So it doesn't mean that if restaurant is using NetSuite, the next restaurant would be using the same set of fields, so things might be different. And that's where you would be adding a custom field.

Now, if you want to access data off of that record, these are the following functions or methods that you're going to use. Well, first, you need to get a reference to the record object, so that's the one thing that you need to get. Now, the second thing that you need to do is that once you have a reference to that record object, then you can use these functions to either get values or set values to the record. For example, if you want to get a reference to the field, you can use the Get Field method.

So the Get Field method would be more of used for UI purposes-- for example, if you want to set the field as mandatory, or if you want to hide a field, things like that. But if you want to get a value, then you're either going to use the Get Value or Get Text method. Now, setting values would be pretty much the same, so you have the Set Value and also the Set Text method.

So if you want to get a reference to that record object from the context object, you can get that from the New Record property. So here, you can just say context or script context, depending on how you defined it, and then dot New Record. So once you get that, the return of this object would be your record object.

Now, there are two properties that you can use-- the new record and also the old record. Now, for our purposes for this particular module, we're going to use the new record property for now, but we're going to distinguish between the two once we start looking at the entry points. But if you want to get a reference-- and mostly people just use the new record property anyway. There are very few people that would use old records, [INAUDIBLE] that has a specific purpose, so getting the recent or the current copy of the record would be from the new record.

Now, regarding logs, we saw one of the logs that are available in NetSuite, which is the debug log. There are actually four log levels that are available. We have debug, audit, error, and emergency, and depending on the kind of log that you want to use, that will be the function that you're going to call, or the method that you're going to call, from your log object. So for example, if you want to make an audit log, you're going to log.audit, or if you're going to make an error log, you're going to use log.error.

Now the question is, what's the difference between the log levels? So the log levels would define the kind of thing that you want to do. Now, the lowest log level is the debug log, so debug logs are typically used if you are still developing your script. So that's the lowest log level.

Now, if you move up a level-- for example, if you want to store information about a certain value, let's say, if you want to create a script where, if a value is edited, you want to log who edited that value, or maybe you're tracking certain values behind the scenes so that you can just go through those logs and then create a report, so you would know how the values have changed in the past, you will be using an audit log. Now, an error log-- this is used if you want to log information about an error. So for example, if your script throws an error, and you want to make sure you inform the developers that an error has been thrown, then you can use the error log. And lastly, here, the highest log level is the emergency log, and the emergency log is basically when there's something really big that happened on your script that can potentially break your script.

Now, the emergency log is not typically used because if there's anything really bad on your script, you need to make sure you handle that, so people would normally just use debug or error. So debug is used a lot because that's what you would use while you're developing your script, and then error is something that you would use if you want to know if something went wrong with your script. So this would be handled exceptions or things that you know that this can potentially be thrown once you run your script.

Now that we know where to use those log levels, it's important for us to understand that these log levels are specifically in this order. So the lowest log level is debug, and then it goes to audit, then error, and then emergency. Now, why is it important for us to know the different log levels and their order? That is because, if you are going to set your deployment records-- so if you remember earlier, we created a deployment record, and then we have different log levels, so debug, audit, error, and emergency-- whatever level that you set on that deployment record, those will be the levels that will be used by the system when you are executing the script.

For example, if I set my log level in my script deployment as debug, which is the lowest log level, that means that the system would execute all log levels that are the same log level or higher. So if I set my deployment to debug, that means that the system would execute all debug logs, all audit logs, all error logs, and all emergency logs. But if I move my log level higher-- for example, on the deployment record, I said that to error-- that means the system would only execute error logs and emergency logs. And that will be very useful if you are a developer because if you've already set your script to release, meaning everybody can execute your script, you really don't want to unnecessarily create logs, so all of the debug logs should be turned off.

Now, if you want to turn off your logs-- I mean, you don't want to go back to your script, search for all of the debug logs, and comment that out. That's going to be a lot of work, especially if you have a lot of log. So the easy way to do that is just go to your deployment record and set that to a higher log level. Maybe instead of using debug, you can just set to audit, error, or emergency.

Now, the system would automatically set your log level to error once you set status to release from testing. So that's something that the system handles for you, but this is also something that you can set. So even if the system set your log level to error once you deploy your script or released, you can still set it back to debug. Again, we don't really recommend that, especially if you're generating a lot of logs. Just make sure you're aware of the log levels and where they're used.

In this walkthrough, what we're going to do is that we're going to expand our script by extracting values of our record. So before we do that, what we're going to do is that we're going to build a custom field, and that custom field would be accessed from our script. We're also going to look at how we can extract values off of other fields, such as the supervisor name, in this case, and also the supervisor ID.

What we want to learn from this module is that, one, we want to learn how we can create a custom field. While creating a custom field is not exactly part of SuiteScript, it is something that you would be doing a lot as a SuiteScript developer. So creating custom field would be part of the SuiteBuilder platform. Also, we want to know how we can get and set values on a record object.

Before building our script, or before expanding our script, the first thing that we want to do is that we want to create our employee code field. Now, the employee code field is a custom entity field. It's a field that we're going to add to our employee record to store information regarding, say, an ID or a specific ID for my employee.

Now, custom fields are things that you can add to your record just in case you want to expand the data that you want to store inside, in this example, this employee record. So here, to create an employee or a custom entity field for the employee record, what you can do is that you can go to Customization, List Records and Fields, go to Entity Fields, and then click on New. So again, that's Customization, List Records and Fields, Entity Fields, and the New.

And then on this page, you're going to define what kind of field do you want to add to your employee record? Now, here, we want to add an employee code field, so I'll just add that-- employee code. And then for the ID, like what I mentioned, it's very important for you to make sure that you use the proper ID naming convention. Now, for this employee code, I'll just use underscore sdr, then underscore. I'll just say employee underscore code.

Now, for the ID, make sure that you use all lowercase, and even if you try to use mixed cases, the system would automatically convert that into all lowercase. So if you want to separate word from another, just make sure you use an underscore. Keep the ID short, though, because you only have a maximum of 28 characters for ID.

Now, the next thing that we have here would be the type-- what kind of field do you want to add to your record? You can add a host of different fields. You can have a checkbox, a currency field, a date, and so on. In this case, we just want to add a free form text, so I'll just keep that as a default. Free form text fields would be similar to this label, where the user can type in any value that they want.

Now, moving on here to the bottom, I have the Applies To subtab, which would tell me, where do I want to save this particular field? So here, I want to apply this to my employee record. If I apply this to other record types-- for example, a contact, a customer, a project-- I can also select those values.

So here, I'll just click on Save so that I can save that to my employee record, and I'll just reuse this page. If I open an employee record, say I'll open-- I think I'll open the same record, Aubrey Pober. If I click on it here, so I can see, where is the field that I've added? If I scroll down, what's going to happen is that, as the list of subtabs, I have a Custom subtab. So if I open my custom subtab, that's where I would find the field that I've added.

Now, one thing that you have to remember, though, especially if you're an administrator-- so if you have a dual role of a SuiteScript developer and also the NetSuite administrator-- is that try to avoid leaving your fields at Custom, because as a user, it doesn't really matter for me if a field is custom or if a field is standard with NetSuite. What's important for me is that I need to know where I can find the information. For example, for employee code, I probably wouldn't look at custom because I don't know if employee code is a standard field or a custom field. You want to make sure that any customization that you create on the system would be user friendly.

So maybe, for employee code, what you can do is that you can put that under Human Resources. So that seems like a logical choice. Or you can put that inside the body field, so the body section would be the fields that are above your subtabs. So for example, let's just put our employee code field maybe right over here, or maybe anywhere above our subtabs.

I can do that by going back to my customization, so if I open my employee code definition here-- so if you remember, this is what we were configuring earlier. If you go to the Display subtab, and then under the Display subtab, you have a field here called Subtab. So the subtab would define where you want to put this particular custom field.

Now, in my case, again, I can put that under Human Resources because employee code-- I think that would be logical if I put that in Human Resources. Or if I want to put that inside the body of my form, I can just click on main. So main is not a subtab, but it would define the actual fields above the subtab.

So I think this is something that we would be using a lot, so I'll just put that in the main section. So I'll click on Save, and after that, I'll go back to my employee page. And if I refresh this page, as you can see, at the very bottom here, you would see our employee code information.

So now here, I'm going to just add a value here. I'll just say EMP001, so that will be my employee code for this particular employee. And I'm going to save this, and then what I also want to do is that I want to extract certain information from my record.

So on my record here, I want to extract, if you remember from our walkthrough definition, we want to get the name of the supervisor, which is Jeff Shen. We also want to get the internal ID of Jeff Shen's record. And then at the bottom here, I also want to get my employee code value. So how would I extract values off of my record?

So to extract values off the record-- let me go back to my code here. And the first thing I'm going to do is that I'm going to comment out this Hello World call because we don't need that anymore. You need to first get a reference to the record that you're saving.

Now, in this case, since I have deployed my record to an employee record, then I know that when I save my employee record, I will be getting a copy of that employee record. Now, to get a copy of that employee record-- so here, I'll just put that inside a variable-- I can get that from my context object. So every time you do something to a record, you would always have that context object.

Now normally, it could be script context. Context really depends on what you have on your account. You can rename this to anything that you want. So for example, if I rename this to Script Context, then I can use the name Script Context. But this context value basically contains information about the execution of your afterSubmit.

Now, for this, if you want to know more about what information you can extract off of your afterSubmit or any function, you can always go to the Help Center. So let's quickly go to the Help Center here. So inside the Help Center, if you go to the SuiteCloud section-- so again, under SuiteCloud, you have the SuiteScript 2.0. And then under SuiteScript 2.0, you have SuiteScript 2.0 API. And what we're interested in, at this point, would be this section-- SuiteScript 2.0 script types and entry points.

Now, we haven't discussed entry points just yet, but for now, if you just want to know what kind of information you're going to get off of an afterSubmit function, you can just go to a user event type definition. So here, I have user event script type. If I expand that, then I would see that I have afterSubmit here. If I open afterSubmit, and if I scroll down, you can see these are the different parameters that I can get. I have the new record parameter, the old record parameter, and then also my type.

Now, for the type, this is something that we already saw when we created our deployment record. So remember, so here, if I look at the values for my type, you can see here that I have different options such as approve or create. So if you remember, you can filter your script so that it'll only execute when the user is creating a record or maybe deleting a record. You have edited. So these are the different events that can be triggered when you save a record.

Now, going back to my afterSubmit here, I can also see that I have two properties. I have the new record property and also the old record property, which would be giving us a record object. So this would be the record module, and then the record object definition. So both of those would give you the same record object.

Now, if they would be giving me the same object type, then what's the difference between new record and old record? New record would give you information about the current state of the record, so for example, if I updated the record, and then I clicked on Save, whatever information that I updated it to, then I will get that on my new record property. So if I tried to load a record, for example, then I tried to change the value, and then I click on Save, and maybe I want to compare the new value, the one that the user changed it to, with the old value that was on the database before it got saved, then I can get that information from old record. So new record, that will be the current state of the record, either if the record was changed by the user, or if the user-- if the value was not changed, then what you have on old record and new record would be the same. But if the value changed, then new record would be the current state, and then old record would be whatever was on the database before it got changed.

So this would be helpful if you want to compare values. Maybe you're tracking or auditing certain values because you want to know who edited this and what value it was changed to and what it was originally. Then you can just load these two objects and try to make a comparison. Maybe just log that information in.

But for us, we just want to extract whatever the current information is, so we're going to get the information from our new record. Going back to our script here, so employee, so again, we're going to get that from our context object, so context.newrecord. So that'll give us our employee record, and once I have my employee record, if I want to extract information from that employee record, then for example, here, I'll just get my employee code value. All I need to do is, from that record object, I can use the Get Value method.

So the Get Value method has one parameter, and that parameter would be the ID or script ID of the record that I want to extract. For example, if I want to extract, say, my employee code value, so here, if I can go back to my form, and then if I open my employee record here, if I click on the field level help, so if I click on that field label, I will get my field level help. And then at the bottom of my field level help, that's where I would get my ID.

So all I need to do is just copy that information, [INAUDIBLE] parameter of my Get Value. So you can use this for either a custom field, a standard field, or any kind of field. Just use Get Value and then pass the ID.

Now, if you are on your account, then you notice that I went to my production account, but I clicked on, say, Supervisor. When I clicked on Supervisor, I did not see that field ID. So what's going on there?

Now, if you don't see that field ID option, that means that the IDs are hidden from this particular instance of the account. Now, to enable that, what you can do is that you can go to your preferences. You highlight the Home icon. Click on Set Preferences. I'll just open this in a new subtab.

Then on the preference settings, make sure that the Show Internal IDs option is turned on. This would be dependent on the users, so some users might have this enabled, and some users might have this disabled. As developers, it's very important for us to make sure that this is enabled because we use script IDs a lot. So again, just make sure that's turned on, and once that's turned on, then you can just get any ID from your record just by clicking on your field level help.

Now, typically, the IDs are the same. For example, currency-- if I open currency, you can see here hat the field ID is also currency. If I click on mobile phone, again, you can see the is mobile phone, but it's not always the case. For example, for my employee ID, if I open Employee ID, you can see here that my field ID is actually Entity ID, so be careful with that. Make sure that you always check the documentation before using an ID.

So that's that. So again, we want to also get the name of my supervisor, Jeff Shen, and also the internal ID. So if I opened this in a new subtab, and if I go to my URL, I can see here that the ID of my record, Jeff Shen, is 13. So I want to get that value.

Sometimes, it's important for us to get the internal ID value so that we can do something to that other information. So in this case, this is another employee record, and if I want to load that employee record, I would be needing my internal ID value. So we want that, and we also want the name that's listed in our form. So how do we get that information?

So going back here on our code, so we want to prepare our data there. So here, we want to get the supervisor name, and then we also want to get the supervisor ID. There we go.

So for the supervisor name, if you want to get the supervisor name, you have to use a different function or a different method. So here, from my employee record, I need to use the Get Text method, and then the parameter would be the same. So basically, I'll just type in the ID. So if you remember when we were looking at the ID earlier, the supervisor script ID is also supervisor, so that'll be Get Text. Now, if I want to get the internal ID, then I would have to use the same Get Value function, so here, employee.getvalue, and then supervisor.

Now, you have to remember that the Get Field and Get Value-- or Get Text and also the Get Value functions or methods, those are used to get different kinds of information off of your record. If you want to get information out of regular fields-- for example, going back to our employee record, if you want to get the employee ID or the name, initials, job title, and so on-- you would use the Get Value method. But if you want to get information off of a list or record field-- now, if you're new to NetSuite, when we say list or record field that would be fields that has dropdowns. So for example, if you have supervisor, you want to get value off of supervisor, image, you also have here department, class location, billing class, and so on-- so again, as long as it has a dropdown, then you can use one of two methods.

You can either use Get Value, or you can use Get Text. The difference between Get Value and Get Text is that, if you use Get Value off of a list or record field, again, anything that has a dropdown, then Get Value would return the internal ID that's related to that list or record. So in this case, if I want to get Jeff Shen's internal ID, then I would have to use Get Value. If I want to get the name or whatever is displayed on the form, then I would have to Get Text, so that's the difference.

Now, Get Text would only be applicable if you're dealing with a list or record field, or any field that has a dropdown. If the field does not have a dropdown-- for example, say, email, phone number, mobile phone, home phone-- again, the phone number is incorrect. Let me just try to fix that.

So for these kinds of fields, then what you would use is Get Value. You're not going to Get Text for this these fields. Get Text would only be applicable if you're dealing with [INAUDIBLE].

So that's it. So now that we have gotten those values-- and let me just fix this a little bit so that it would look right there-- the next thing we're going to do is that we're just going to log the information at log.debug. And then for my debug here, I'll just say employee code. Then on my employee code-- and by the way, my log.debug function has two parameters. That would be the title and also the details.

So we're going to look at that later and where we can find those information, and I'll just copy that and then create one for my supervisor ID-- supervisor, if I can spell that right. Supervisor ID, and also my supervisor name. Of course, change this so that we're going to get the right value-- supervisor. So this is my supervisor ID, and then I have my supervisor name. There we go.

So that's it. We're done. So again, in this walkthrough, what we want to learn is that we want to know about the different functions that you can use to get a value. Now, setting a value would be pretty much the same, so if you want to set the value, then you're going to use either set value or set text.

Now, with Get Value, you only have one parameter, which is the name of the field that you want to get. But if you want to set a value, you have two parameters. The first parameter would be the value that you want to set, and the second parameter would be the value that you want to set it to.

So for example, if I want to change my employee code-- so just as an example, here, I can just say employee.setvalue, and then for my employee code, I can copy this. Put that right over here. And then for my value, if I want to change that to maybe EMP002, I can just have this definition. So that's something I can do.

So let me upload this, and there you go. Looks like it's already uploaded, and if I go to my script record-- let's just use this tab to open my script record. There we go. We open this, and let me go to my execution log subtab.

So you can see, it's still my Hello World script, but if I save this record, that should trigger my afterSubmit again. And if I go back here and then click on Refresh, as you can see, we were able to get the information off of our record.

So remember the log option-- so you have two properties here for log. So you have an employee code, so that will be the first parameter for your log. And then the second parameter would be the details.

So for the title, or the first parameter of your log, then what you have here is the name of your-- or kind of like an initial marker for your log. Now, there is a limitation on the number of characters that you can use on the name, but you have a larger set of characters for the detail. So if you want to log more information, such as maybe longer pieces of text, make sure you put that on the details instead of the title.

So again, these are the information that we've gotten. We have our supervisor ID, so that is 13. And then you also have your supervisor name, and then the name of the supervisor here is Jeff Shen.

Now, regarding the log that we have, if I go back to my script here-- now, the option that we used here is log.debug. Now, earlier, we saw that we have four different log options. So if I go to My Account again, if I go to my deployments-- let me just edit this. So you can see here that I have four different log levels, so I have debug, audit, error, and emergency. So for this, this defines what am I going to-- what kind of law do I want to do?

Now, this is the actual hierarchy of your log levels, so you have debug as the lowest log level. You have your audit, you have error, and then you also have emergency, which is the highest log level. Now, if you set your log to debug, that means that everything in this level, debug, and higher, which is audit, error, and emergency, would be logged. But if I change that to say audit, that means it'll get whatever's in this level, audit, and then higher, which means audit, error, and emergency. And it would ignore your debug logs.

This would be very convenient if you want to disable logging for your account because you don't want to generate too many logs. If you generate more than 100,000 logs per 60 minute time period, what's going to happen is that the system would automatically send the owner of the script-- so in this case, that'll be me. I will get a notification saying that this script is generating too many logs, and then the log level would be moved from the current log level to the highest-- or not the highest, but the next higher log level.

So if the log level was originally set to debug, it will be moved to audit. If you're generating too many logs in audit, then it will be moved to error. And then if it's generating too many logs in error, it'll be moved to emergency. So that's what you're going to get.

Now, so in this case, if I want to disable my logs, what I can do is that I can just set that to error. By the way, for the logs-- debug, that's something that you use for debugging. Audit, if you want to log certain information, or if you want to check whatever is happening on the script execution. Say, if you want to see if a certain section of the script was triggered, or maybe if you want to log pieces of information from your record, that's something that you can do.

If you want to log only error messages, you can just set that to error. And then emergency would be if there are any kinds of error that can potentially affect your code adversely-- for example, it can break your code, or do something that is-- or maybe an error that can possibly cause the project to fail, or something like that. So for example here, if I move the log level, say, to error, click on OK, then I will save this script record.

And if I edit-- and what I can do here is actually I can just remove all of the logs so that we can see that even if we trigger my afterSubmit event again, if I go to my script record, and if I click on Refresh, you can see the logs were not triggered because on our script, the logs were set to debug. And then since the log level that we selected, error, has a higher log level, anything below that would be ignored. So any audit logs and also any debug logs would be ignored.

Again, very convenient so that you don't have to go back to your code and comment all of your logging. Anyway, so let me just set that back to-- so here, I'll just change that back to debug because we will be using debug logs quite a bit in the future. So that's about [INAUDIBLE].

So just a review of the functions that we have here-- so context.newrecord if you want to get the record reference, and that, again, would depend on the record that you are trying to load. Record.getvalue if you want to get the value or the ID, depending on the field that you want to get. You're going to get an ID using Get Value if you are getting a value off of a dropdown field. Set Value if you want to set that value.

Get Text only used for list or record field or anything that has a dropdown. That's what we call the dropdown fields-- list or record fields. So you would get the name that's written on the actual field. Set Text if you want to set a value of a dropdown, and if you want to perform debug logs, you can use log.debug, or if you want to write error logs, that will be log.error.

Before we move on to the next section of our walkthrough, it's important for us to understand a very important concept in SuiteScript development, and that would be accessing the document object model. If you are an experienced JavaScript developer, this is something that you have probably accessed in the past. Every time you want to extract certain information on the form, what you can do is that you can get a document object, and then inside your document object, you can extract certain pieces of it by just using this hierarchy.

So for example here, you can see that I have a document object, and that inside my document object, I have a form. The form is named Form 1, and then inside that form, I have several fields. And if I want to set the value of Field 1, I can set field1.value and then equal, and then whatever value that I want to set. Now, this is something that you can also do in SuiteScript, but this is highly discouraged. Why do we discourage using the document object model?

Now, if you want to set a value or get a value off of a field, what you want to do is that you want to get that using the record object that you have gotten from your context object. If you used to Get Value, Get Text, or if you want to set the value, Set Value and Set Text methods, then that would also perform a lot of customizations in the background. So for example, it has browser checks to make sure that it would be working regardless of the browser you're using. There are some optimizations to make sure that the information is always up to date on the server, and things like that. But if you use the document object model, you're missing out on all of those information, so you want to make sure that you always use either the Get Value, Get Text, or Set Value or Set Text method.

Also, DOM is not futureproof because this would be based on the name that the UI Designer have used on that field. So for example, if you want to act a form, it might not be Form 1 on another version of NetSuite because if we change the UI design-- for example, I have my current design of NetSuite, and then NetSuite decides to upgrade the design and then change the names that are being used in the background, then the script that you have created that accesses the document object model will break. So DOM is not futureproof, so you want to make sure that, instead of using the document object model, you would always just use your Get Text, Set Value, and Set Text method.

This is very, very important. Again even if you use Get Value or Set Value, you would always just use the script ID, and those script IDs would never change in NetSuite. The elements in the background might change if the UI changes, but the IDs would have been always like that regardless of the upgrades done in the system.

Now, let's move on to something that's also important for us for SuiteScript development, and that would be debugging. Debugging in SuiteScript is a little bit different because, unlike other traditional SuiteScript APIs, where scripts are only executed on the browser, you have an option in NetSuite to either execute your script either on the client side or on the server side. So in this module, we're going to look at how we can debug server side scripts. In future modules, we're also going to look at how you can debug a client side script.

There are a few things that you have to remember when you're debugging. Now, for this debugging session, we're going to look at server deployed scripts. Now, for this walkthrough, I'm going to show you how you can debug user event scripts, but you can debug any kind of server side script using the steps that I'm going to show you.

Now, before we start debugging scripts, there is a very important thing that you have to remember, and that would be where you're going to debug your script. Whenever you're debugging, you have to log into a separate web application server. Currently, the server that we are logged in is the production server.

Even if we're using a training account, it's still running on a production server. So here, I have system.na1.netsuite.com, which is the data center for the NA Northwest data center. So if you want to debug my script, I have to log into this particular data center, so debugger.na2.netsuite.com.

Now, the debugger application server actually runs a little bit slower than your production application server, and it was designed that way so that you can easily stop the debugging session or can capture actions on the debugging session so that you can look at what's happening to your script. So if you are running or testing or developing your script, and you don't need to debug, I do recommend that you do that in the production server because that will be faster. But if you do need to debug, you need to go to the debugger application server.

Now, even if you have two application servers, please do note that it would still be using the same database. One of the common misconception is that I'm logging into another account or another application server, maybe I don't really need to worry about the database, but that's really not the case. So anything that you would do on the debugger server, that would also affect your production server because, again, they are using the same database.

You have to make sure that you be careful with anything that you do on the debugger application server. Again, they are sharing the database. Very, very important.

In this walkthrough, what we're going to do is that we're going to debug our user event script. Now, we are going to debug a user event script, but you can apply the same technique to any kind of server side script in NetSuite. So what we want to learn here is that we want to learn how we can log into the script debugger, how we can debug a user event script, or any kind of server script, and then we also want to look at the tools that are available for using the debugger.

So the first thing that you need to remember if you're going to debug a service side script is to log on to the script debugger. You can do that by going to Customization, Scripting, and then Script Debugger. So again, that's Customization, Scripting, and then Script Debugger.

So if you open that link, you will see here, you have a description of the script debugger, and all you need to do is just click on this hyperlink that says Click Here to Log On to the Script Debugger Domain. So once I click here, I will be moved from my production domain or production web application server to the debugger application server. So I can click on that. Here, I'm going to be asked to log in, and you know that you're going to log into the right domain if you look at the URL. It should say debugger.na2.nesuite.com.

So let me go ahead and log in, and as soon as I log in, I would be given the script debugger page. So in the script debugger page, you have two options for debugging your script. You can either debug an ad hoc script. So you can type your script here, so if I want to build a script, or if I have a script, so I can just type that information here. Or if I have an existing script that I've already deployed to the system, I can just click on Debug Existing, and that's what we're going to do since we've already deployed our script.

So to debug your script, just click on Debug Existing. You will be given an option here to select the script that you want to debug. In our case, we only have one script that we can debug, so I'm going to select that, and then click on Select and Close.

Now, as soon as I click on Select and Close, notice here that it says Waiting for User Action because it still has to be triggered the normal way for the script debugger to work. So in this case, since our function is stored on the afterSubmit, then I have to make sure I save a record to trigger this. Now, I have to do this quickly because the debugger would automatically expire if I take too long.

So anyway, let's see-- let me wait for that load and then open Aubrey Pober's account. And then I'm going to save this record. Then after saving this record, go back to your script debugger page, and as you can see here in my Script Debugger, [INAUDIBLE] afterSubmit function was caught by my debugger. So you can see I have a script here.

And if you want to go through your script, you have the normal debugging options, so you have your Step Over if you want to go to the next line of your script. You also have an option here to Step Into, so just in case you have multiple functions inside your module, and you want to go inside the function that you're calling, then you can click on Step Into. If you want to go back to your main function, click on Step Out. You also have your Continue button if you want to move your execution and then stop at a particular break point. Or if you don't have a breakpoint, then it will just execute the whole script.

For example, let's try this, and let's say that I add a breakpoint. And by the way, you can add a breakpoint by clicking on the empty space beside your line number, so that will be your breakpoint. So if I click on Continue, as you can see, it'll just go ahead and run my execution and stop at that particular breakpoint. Again, if I don't have a breakpoint, then it'll just complete the execution.

Then aside from those functions, you also have some subtabs here at the bottom. You have the execution log. This is similar to the execution log that you would see inside your script record, so if you have any logs, like a debug log, it will display that in your execution log.

You'll also have local variables, which would give you all the variables for this script executions. For example, you have your context object. You also have your employee object, so we are trying to extract that employee object from our execution. Looks like our script has finished executing here, so let's go ahead and try this again. Let me refresh my form.

Debug existing. Let me just refresh my execution here. Save and Close. Again, be careful with debugging because-- or make sure that you are not spending too much time, and I am spending too much time explaining what's happening. So that's why I timed out.

So anyway, going back to the local variables here, so you have your employee object. So this is what we're extracting from our context.newrecord. We also have our employee code, so you can see that the employee code is EMP001. I have my supervisor name, which is Jeff Shen, and then I also have my Supervisor ID. And if I want to continue with the execution-- for example, if I go to the next line, if I click on Step Over, if I look at my execution logs, you can see that I am logging my employee code and then the details, and I'm logging [INAUDIBLE] the value, which is EMP001.

Now, if you have several variables, like maybe you have a very long list of variables because you have a long script module, and you want to watch a certain expression or watch a certain value, you can go to Watches and then say, I want to look for my employee code. So that instead of looking for that employee code value here, if you have maybe dozens of variables listed, you can just go to Watches and find the exact information that you need. You also have your evaluate expressions, so you can use this if you want to check certain values, maybe perform an ad hoc computation, and you do whatever you want with the record. So for example, if you want to see what's going to happen if I add this particular value, or if I change the value to something else, you can put that here.

So if I want to say, what's the value of a field that I have and extract it on my script, say phone number, I can do that in my evaluate expression. So I have employee.getvalue and then I have phone number. So phone would be what I want to extract here, so I can see that the phone number is 555-5555. Now, you're not limited to just getting value. Again, anything that you can run on your script, you can also run on your evaluate expression.

And lastly here, you have your breakpoints. So if you want to list all of the breakpoints that you have, you will be able to find that here. Or if you want to remove a certain breakpoint-- let's say, I have this one breakpoint, and if I want to remove that breakpoint, I can click on the x icon here, or I can also click on this. It's really up to you in how you want to do this.

The Breakpoints subtab would be very helpful if you have several breakpoints, so instead of looking for the particular breakpoint that you want to disable, you can just go to the list of breakpoints, and then you can disable here. So for example, if I want to remove this breakpoint, I can just click on that, and as you can see, the breakpoint disappeared. And if I click on Continue, that will just continue the execution.

So if I go to my execution logs subtab, you can see that I was able to get all of the information. So this is something that you will be using if you are going to debug your server side script. Now, if you are going to debug a client side script, you are going to use the debugger that is built into your browser, and we're going to talk about that in the future module.

Now, one last thing that I want to mention before we end this walkthrough is that you need to make sure that your script is properly set if you want to perform debugging, and there are two things that you need to remember with your script settings. So if I go to Scripting, Script Definition, so I'll just open this in edit mode, the first thing that I need to remember is that I can only debug scripts that I own. So if the script belongs to somebody else, then I would not be able to debug it. So if I change this to, say, Luigi, click on Save, and if I go to my script debugger, if I click on debug existing, the script that we were able to debug [INAUDIBLE] because the script does not belong to me, or I'm not the owner of the script.

Also, another thing that you have to remember is that if you are debugging a script, you can only debug scripts if the script is set. So here, I'm going to change this back to Larry Nelson, and then on the deployment, I can only debug scripts that are set to testing. If I set my status to released, that means that this is already in production, and that's not something that I can debug. I can only debug scripts that are set to testing. So if I click on OK here, click on Save, again, if I go back to my debugger, click on debug existing, you'd be able to see here that the script is not listed.

So if I close this, set those two settings back, make sure that the owner of the script is me-- so in this case, it's Larry Nelson-- and also my script status is set to testing, if I save that, and if I go back to my script debugger, if I click on debug existing, as you can see, I can now debug my script. So please remember that. Make sure that you are debugging, those two options are properly set.

So that's about it for this walkthrough. Some things to remember before you go to your exercise-- so one, make sure that you understand the difference between Get Value and Get Text. If you are going to get values off of any of the regular fields or any fields that does not have a dropdown, you would always use the Get Value method.

Now, if you are going to get information off of a list or record field, and list or record field would be fields that has a dropdown, then you can possibly use these functions-- Get Value and Get Text. Get Value would give you the internal ID of the field, and Get Text would give you the text that actually appears on the field. So if you use that for supervisor, in our case, then Get Value would give you the internal ID of that supervisor value, and Get Text would give you the value that's written on the field. So for example, for Jeff Shen, it's going to give you the string Jeff Shen.

Also, like what I mentioned, avoid using the document object model. So please make sure that you use to Get Value, Get Text, Set Value, and Set Text method so that your code would be futureproof, because if you use the document object model, and the UI change, and the background or the underlying structure change a little bit, then your script will break. So you want to avoid that, and you want to make sure that your script would be working even if we move to a different version.

Also, it's very important for you to use the proper ID naming convention, and it's not just you. You need to make sure that you inform the rest of the company or anybody that has access to any kind of customization to NetSuite, so that every time they see an ID field, they don't leave that blank but give a proper name for that ID. It'll make your life so much easier if the IDs are properly named.

And also lastly, you are going to, instead of using the DEFINE statement for debugging ad hoc scripts, you are going to use the REQUIRE statement. Now, what is the difference between the DEFINE statement and the REQUIRE statement? The structure is a little bit different, but the general syntax is the same. So first, if you have a DEFINE statement, you can just replace the word DEFINE with REQUIRE, and then just remove the RETURN statement. That's it, but everything else would be generally the same.

We are going to look at how we can use this once we get to the searching module, because that would be perfect for, as an example, for debugging ad hoc scripts. Now, it's your turn. Please complete the following activity. This concludes module 3, Using SuiteScript Objects.