Module 14, Important Considerations. In this module, we're just going to review some of the considerations that you want to think about if you are going to be a full fledged SuiteScript developer. So in this module, we're going to look at what the recommended development process is.

So we're going to look at how you're going to develop if you have a sandbox account, or how you're going to develop if you don't have a sandbox account. Also, we're going to look at how you can handle SuiteScript errors. So that's something that you need to know.

Also, we're going to talk about governance. So I've mentioned governance a few times, but we're going to try to understand what governance really is, and why it is important for us to know that.

We're also going to look at the APM, or the Application Performance Management tool, so that we can check how is our record performing. Are we just dumping too many scripts from that record? Is it slowing things down? So we need to know that. And also we're going to look at how we can package up all of our customizations using SuiteBundler.

Let's start with our first topic. Let's talk about the SuiteScript development cycle. So this is a recommended development cycle that we want you to consider if you are going to develop SuiteScripts. Now, we've broken this down into two sections. One thing if you have a production account. And then another cycle if you are going to develop on a sandbox account.

Now, during development, what you need to do is you need to make sure that you prepare your test data. So on your account, you need to make sure that you have a set of data that you can call the test data before you even start developing. If you're developing for your own customers, and this is something that I have personally experienced, where customers refuse to purchase their own sandbox account, and they would prefer to develop directly on the production account.

Now, as a developer, that's something that I don't really recommend, because the risk of you destroying your data would be very high. And in fact, that's something that we have seen. We've seen people pay consultants or partners so that they could fix the data that they have accidentally deleted. So that's something that can happen to you, if you don't use a sandbox account.

Now, as a developer, it's important for us to ask for that test data. So you need to-- if you can, for example, if you're a partner, and you're developing over your customer's production account, if you can ask that in writing. Just make sure to tell them that, OK, this is test data. I can do whatever I want to do with this data. Because this would not affect your day-to-day business.

So you need to make sure that you have that set up. And I can't stress this enough. This is very, very important as a developer to make sure that you have test data, and you're actually manipulating test data. Because if you're not dealing with test data, and you're doing your development on production, that's a very bad thing.

So anyway, so once you have your test data, the next thing that you need to do is, of course, you need to develop your script. So once you start writing your script, so set your account to, or set your script to testing, so that it's not executed accidentally by other users.

Now, if you are going to develop on a production account, be careful on when you move your script from testing to release. So we're going to talk about that more. But anyway, so start developing your script. And once you're confident that your script is ready for prime time, that you can release that publicly, don't release your script just yet.

So the thing that you need to do, and this is something that I have seen happen a lot of times, is that people tend to just test, or developers tend to just test their scripts on the administrative role. Now, developing your scripts on administrator role is fine. But testing on administrator role is not really a good idea.

What you want to do is that if you're testing for a particular kind of end user, you have to test using the end user's roles. Remember that on the administrator role, you have access to everything, and your script would be able to run smoothly.

If you use your end user's roles, they might not have access to fields that you have access to. So I've seen this a lot. Like developers would say, OK, this script is done. Go ahead and try to deploy that.

When they run that through the system, the users complain that this is not working. I can't do my job, because this is blocking-- like I need to create employee records, but I can't create employee records, because the script is preventing me. I'm getting error messages. What am I going to do?

And they can't just stop the business because your script is not working. You have to make sure you fix that immediately. And that will not happen if you actually test using the end user's roles. That's something that is very important. And this is going to save you a lot of heartaches if you actually test using the end user's roles. Very, very important.

Next thing that you need to do is you need to also set the log levels. Remember the log levels that we talked about. If you are going to develop your script, you normally would just use the debug log level. And then when you are going to move your script to production, what do you want to do is that you want to change that from, say, debug to either audit or error.

Now, if you have any audit logs, if you're tracking certain information on fields, or any information at all, and you need those logs in the execution logs, then what you need to do is that you need to just set that to audit. That's fine. But if you don't have any audit logs, then just move that to emergency.

Common log levels that are used it's just mainly debug and error. So those are the two logs that you want to use. Remember that if you are generating too many logs, the system will send the owner of the script a message, and the log level would automatically be set to a higher log level.

For example, if you're on a debug log, and you generate more than 100,000 logs on a 60-minute period, then the system would automatically move that log to a higher level, which is audit, and then send the owner a message saying that, OK, this is something that we did because your script is generating too many logs.

Now, after you have this, then after you have configured your script record, then you think everything is OK, the next thing that you're going to do is you need to release. But don't release publicly. Don't just think, OK, I'm going to go to my deployment records, set from testing to release, and boom, that's it.

But no, that's not what you should do. What you should do is that you need to release that to a limited audience. Remember that you are developing on a production account. Since you're developing on a production account, all users on the account who may not be aware that you have created a script will be executing your script.

And you do not want that to happen. So what do you want to do is that you want to release that to certain audiences. Now, remember earlier there was a question about the audience's sub-tab in your deployment record. That is what that's about. So if you want to release that only to a certain set of users, then just release that to a certain set of users.

Like maybe select a certain role. OK, these are the roles who will be testing your script. Or maybe even you can make more precise selection, like these are the users who would be executing your script. So that's what you want to do. You want to get a test period to see that this is going to be used in production for a while.

Maybe, it really depends on your cycle. If you want to have like an end user testing of maybe two weeks or a month. And again, it's just really up to you if you want wanted like longer or shorter. Now, after everything has been tested, like OK, the user has been using this for two weeks and everything is running properly, then that's when you move on to releasing your account, or releasing your script publicly.

Once you release your script publicly, that means that everybody on account would be able to trigger your script. But, now that you have tested using the proper roles, you have changed all of that settings, the end user, or at least a handful of end users have already tested your script, it shouldn't be a problem to release that publicly.

So that's what you need to do. So a few things to remember here, if you are going to develop on a production account. The first thing is that make sure that you test, or make sure that you have test data. So that's one thing that's very important.

Second thing that's also important is make sure that you test using the end user's role. And then, the other thing that's important here is to make sure that you only release that to limited audiences. So just remember that if you're dealing with a production account.

Again, this is important because you are developing on a production account. If it's a sandbox account, then it's a very different story. On a sandbox account, you are free to do whatever you want. Just go ahead and start developing your script. It does not matter, because you cannot break a sandbox account.

If you happen to, just in case, accidentally break the sandbox account, that's fine, because you can issue a refresh. Now, for those who are not familiar with the sandbox account, a sandbox account is a copy of your production account. But it is housed on a separate database.

For example, if you have a production account, you can get the actual data. And that's the beauty of having a sandbox account. That's a copy of the actual data on your production, meaning those are real values that you can manipulate, and you don't have to worry about messing things up, because that's on a separate database.

Even if you accidentally delete everything, you can issue a refresh to get all of the things on your production account back to your sandbox account. So that's something that you might want to consider. And developing directly on a sandbox account is so much less stressful than developing directly on production.

Now, after you have developed your script, same thing. You need to make sure you test using the end user's roles. That will not change, because you still need to check the permissions of your end users. Again, you can develop using the administrator account, but if you are going to test, make sure to test using the end user's rolls.

Also, make sure to set the log level ones you're done. Set that either to audit or error, depending on what you want to do. Also, here in the sandbox account, you are free to release this to a full audience.

Why is it OK to release that to a full audience? Because not everybody will have access to the sandbox account. And even if they do have access to the sandbox account, they most likely will not be logging into the sandbox account unless they are doing something specific, such as testing your script.

So if you release your script publicly, this would be the same settings that you will be using as if you're going to use that in production. And again, you don't have to worry, because you only have a limited set of users on a sandbox account.

Now, once everything is tested, for example, you've asked your end users to go to the sandbox account and do testing for a certain period of time, then you have to transfer your account, or transfer your customizations, your script, workflows, and everything from your sandbox account to your production account. And you can do that using the SuiteBundler tool.

For the SuiteBundler tool, we're going to talk about that later. But basically, it allows you to gather all of the customizations. You're not limited to scripts like everything, like if you have like save searches, forms, records, fields. Anything that you want to package up, you can package up using the SuiteBundler tool, and then transfer that to your production account.

And of course, on your production account, you will be picking that bundle up. So you're going to install the bundle. Then once you install that bundle, you can do a quick smoke test. But since you have tested on a sandbox account, which is, again, a copy of your production account, that normally should work.

Now, just to make sure, I would still recommend a short-term end user testing. Because sometimes, if you are on a sandbox account, there are a lot of things that you might have already modified that you haven't really modified on the production account.

The behavior might be a little bit different depending on what you've done on your sandbox account. So you have to be careful. So I still recommend that after you install the bundle on production, just have a short term end user testing.

Now, next up let's talk about error handling. So how are you going to handle errors in NetSuite? Now, error handling in NetSuite is pretty much the same as error handling in JavaScript, because we are going to use normal exception objects in SuiteScript.

So to handle an exception in SuiteScript, you have to use the try and the catch block. Again, this is the same. Now, how you're going to manage your exceptions would be up to you. You can have a complex exception handling system, where you have your own set of errors, depending on the script that you're creating, or you can just stick with whatever error objects that are available in NetSuite. It's really up to you.

So let's try to look at an example of an error handling code. So here, we have a try and catch block, and we have a call to the record module, trying to load the record. So as you can see here, I have a type that is incorrect. So I have a record.Type.EMPLOYEEx.

So that means that this would be an invalid value. And of course, my internal ID would also be invalid. So I'm going to have an error. So every time you get an error, or if you think that any parts of your code will produce an error, make sure you put that inside the try and catch block. So loading records, calling web services, things like that, you need to make sure that you add a try and catch block.

Now, on your catch block, the thing that you have to remember here is that error messages in NetSuite, or exception objects in NetSuite are converted into a JSON string. So for you to be able to process that error message, you have to convert that into a JavaScript object, or a POJO, a plain old JavaScript object.

You can easily do without using the JSON.parse function. So this is something that we've already used. And now that you've gotten that, there are three different error messages that you can possibly get. One would be a normal exception. So this would be your standard JavaScript exception object.

And on your exception object, you have these two properties, and also your stack trace, of course. You have your name, so that's part of your exception, and then your message. Again, it's always going to have the name and the message, because that's part of all JavaScript exception objects.

Now, if you are going to get additional exceptions, or like NetSuite specific error objects, there are two kinds of objects that you can get. One would be a SuiteScript error. So that's one kind of object. And then the other one would be a user event error. Now, what is the difference between the SuiteScript error and the user event error?

The SuiteScript error would happen if you have an exception during the execution of a script that is not inside a user event script. So for example, if you're trying to load a record, and then that record that you're loading is incorrect, like you've used an incorrect ID, or if you've used an incorrect type, then that will throw a SuiteScript error.

So here, in that SuiteScript error, you can get these properties. These are just examples, by the way. If you want to see all of the properties that you can get off of your SuiteScript error and user event error objects, you can go to the Help Center.

So if you're on the Help Center, open the module definition for error-- so n slash error. So if you open that, you would see the two types of errors, which is the SuiteScript and also the user event error. So I do recommend that you display the cost and the stack trace. That would help you a lot in troubleshooting.

Then the last error that we have here is the user event error. For the user event error, that would be an error that will happen on a user event script. And this would give you additional information, such as where or what kind of event, or what kind of trigger was the user using, or what event was there when the error occurred.

For example, was the user creating a record? Was the user approving a record? Was the user printing? Or something like that. So those additional information might also help you with troubleshooting.

Now, if you are going to handle error messages in SuiteScript, you have to make sure that you distinguish between the different error messages. Now, for normal error messages, again, you have both error and message. But there are some additional properties that you can get for other objects.

So if you don't distinguish-- for example, if you just have the exception.event type, and it's a SuiteScript error, you're going to get another error inside your catch phrase, which is like throwing another error inside an error handling. So it's not really a good idea.

So if you want to distinguish between the different errors, there is a type property that you can use. So that's the exception.type. Then for the type property, just put in the name of your error object. So for example, for the SuiteScript error, that's error.SuiteScriptError. And for your user event error, that's error.UserEventError.

So just remember that. Again, this is not exactly different from doing any kind of error handling with other JavaScript APIs, because this is still JavaScript. It's still going to be a try and catch block. It's still going to just display the message and also the stack trace. So you just get more information regarding SuiteScript in the NetSuite objects.

Now, if you want to create your own error objects, like OK, you know what, yeah, it's really nice that the NetSuite has its own objects, but I have a very complex application that I'm building, and I want to know exactly what's going to happen to my script. Like if this breaks, I want the user to do this exactly.

So if that's what you want to do, what you can do is that you can create your own error. So there is a method inside our error module, so that's error.create, where you can have your own name and your own message for your exception.

This would be very useful if you, again, if you want to deal with a more complex error management system, where you have a certain naming convention for the error, and then you can have like support look at that particular error. Let's say you have a gigantic project, and then if the user encounters this error, it's like, OK, I'm getting ERR07952. And then this is the error.

So that person can just send an email to the developer and say, OK, this is the error. And you can look at your list and say, OK, that error pertains to this, and this is how you fix that. So it'll be easier for you to track, in case you have that error management system.

But again, you don't need to do that. But you can if you want to. So for example, for partners who are selling their application to customers who have maybe a complex application, or maybe if you are a customer, and then you're developing a project from the ground up, where you want a very complex automation system in NetSuite using SuiteScript, then try to consider creating your own error system.

But if you are going to create your own error system, you have to manage that on your own. So you need to have your own documentation, and you can just manage that on your own.

So here, for example, I have my error object. So once I've created that, what I can do is that I can throw that. So this is just a simple throw keyword. This is native JavaScript. You can throw an object, if you already have an error object, or you can throw a string-- you just want to display a message on the screen. And as always, if you want to handle your error messages, you can just use try and catch block.

Another important topic that we need to talk about would be SuiteScript governance. So what is SuiteScript governance? SuiteScript governance is a way for NetSuite to monitor and check performance of your scripts, to make sure that everything is running properly, and that no script is hogging all of the resources.

Remember, most of the scripts that you will be creating would be on the server side, and you can potentially eat up all of the resources if you improperly program your script. So that's what SuiteScript governance is about. Now, how do we implement SuiteScript governance? So let's try to take a look at that.

So this is an example of a user event script. And as you can see on the user event script, we have some labels here where a user event script is allowed 1,000 units per execution. So every time you execute your script, you are given 1,000 units. And every time you perform certain functions, such as in this case when I'm loading an employee record, I use up 5 units of those 1,000 units.

Whenever I save my record, I would be using 10 units. Now, to better understand this, let's try to look at the Help Center, and look at the documentation for SuiteScript governance.

So to know more about SuiteScript governance, all we need to do is just look for SuiteScript governance in the Help Center. Now, once we open the SuiteScript governance page, so we see here there are two articles that are important for us to look at.

We have the API governance, and also the script usage units limits. Let's try to open the second article first. Now, once we open this article, we see these are the different script types, and the number of allowed units per execution of the script.

For example, a scheduled script would have 10,000 units. User event script, 1,000. Client side script, 1,000. Suitelet is also 1,000. RESTlet, 5,000. Portlets, 1,000. Bundle installation script is 10,000. And then workflow action scripts would be 1,000. What does that mean?

Let me just go back to the article first so that we can have an example here. So I'm going to go to the API governance article. So here, every time we execute a certain function, it would use up a certain number of units. Now, these are SuiteScript 1.0 scripts, but this is still a good article to use as a reference.

We're going to look at some of the SuiteScript 2.0 API so we can find more information about how many units they consume. But again, this would be a really good reference, because it lists all of the functions that can potentially use units.

But anyway, so going here on our list, if we want to load a record, and that'll be nlapiLoadRecord, if you're dealing with SuiteScript 1.0. That means that every time you use nlapiLoadRecord, it'll deduct two units if you're using custom records, or if you're using a non-transaction record, like maybe an entity record, or a CRM record, or an item record, so that'll be five units. Or if you're using a transaction record, that'll deduct 10 units on your execution.

Now, for example, if I'm creating, say, an expense report, so I'm loading an expense report, so that'll use up 10 units. And if I save my expense report-- let's see, I have nlapiSubmitRecord, so that'll be equivalent to my save. That will be using up 30 units total, because 20 units for saving, 10 units for uploading, or loading.

Now, if I try to compute for that, if I'm opening this on a user event script, so a user event script has 1,000 records. And if I divide that by 30, because that's the amount of units that I'm going to use up, that means that I can only update 33 records.

So you have to be careful, because if I don't compute for the number of units, what can happen is that your script can break, and you will get an error message saying that, OK, you have exceeded your governance usage. And that would stop the execution of your script. And you don't want that to happen.

Now, remember that the number of units will be replenished every time you execute your script. So for example, if you put a script on a user event script on an after submit event, then what's going to happen is that every time the user saves the record, that'll trigger another refresh. So that's 1,000 units every time it's saved.

It's not so bad for user event scripts, but if you're dealing with large amounts of records, then it could potentially be a problem. So again, trying to make sure you remember that. Know the number of units that is available per script. And know how much your function would consume.

Now, looking at the documentation for SuiteScript 2.0, let's say if I load a record-- and it should be the same between SuiteScript 1.0, and SuiteScript 2.0. So let me open my record.load definition. And as you can see here as part of the documentation, you have some governance units.

So I have my transaction record. Then unit. So pretty much the same. Other records. That would be non-transaction records such as entity records, item records, and so on. So that's five units. And custom records would be two units. So this is where you're going to get the information. So every time you perform a function, again, just know how many units that's going to consume.

You might find that the governance limitation very limiting. At least for new developers, that's what they find, because they're accustomed to just creating a script. And if something goes wrong with the script-- for example, if the browser freezes because the script just went on an infinite loop, they can just close the browser window, and then all is fine.

But again, remember that this is running on the server. You can't just close the browser and then hope that everything would be OK. You actually have to stop that execution on the server. And this is NetSuite's way of preventing that.

But there are a lot of things that you can do to work around that limitation, or to optimize your code so that you won't hit that limitation. One example here is that instead of using the record.load and record.save combination, what you can do is that you can just use the record.submit fields method instead.

Because if you use the record.load and records.save combination, that would use up an the transaction records 30 units, but the record.submit submit fields, on the transaction record-- or actually, it will just use, I believe, 10 units. Again, you can confirm that by going to the documentation. Similar thing with your results set.

Like what I've mentioned previously when we were talking about searches, if you use the results set instead of a page range, what's going to happen is that it's going to give you 10 units, or it'll consume 10 units per execution. If you use the search.lookupFields, this would only give you one unit per invocation.

So as you can see here, if you use the ResultSet.each then versus one on lookupFields, so you can process 10 times more records if you use search.lookupFields. Or what you can also do is that you can offload your processing to another script.

So if you're running on a scheduled script, what you can do is you can push the processing to a scheduled script, which would give you not just 1,000 records, but 10,000 records. Or better yet, what you can do is that you can use the map/reduce script, which doesn't really-- or which effectively handles governance by automatically rescheduling as necessary.

So those are just small things that you can consider whenever you're thinking about governance. But the main point here is that you need to make sure you think about governance. Because if you don't think about governance, and you do hit that governance wall, then it would delay your deployment, or it would delay your project, because you have to refactor your code to make sure that that does not happen.

So again, if you think of this, or if you tried to fix this after the fact, then it can potentially significantly delay your project. Whereas if you think about this before you actually start coding, then it can really save you some time. So yeah, just do please remember that regarding governance.

Now let's talk about the Application Performance Management. The Application Performance Management tool is used so that we can optimize our code, and gives us information on how our NetSuite account is performing in general. So let's take a look at what information we can get from this tool.

Here's an example of the APM, or the Application Performance Management dashboard. With this dashboard, it gives you a list or tiles of the records that you might or might not have customized. So if you are going to get more information about a particular record, you can see, OK, maybe this record is high volume, and it's costing a lot of slowdowns because of the customization, or if another record you can verify that, OK, it looks like this record has a lot of customizations, but it looks like there's no really significant slowdown.

And again, the information that you can get from the Application Performance Management would help you optimize your code. Now, if you want to install the Application Performance Management, there is a link in the Help Center so that you can find the bundle that you can use to install the APM.

Just look for that bundle on SuiteBundler, and then once you find that bundle in SuiteBundler, just go ahead and install. Now, about SuiteBundler, we're going to talk about that next after this.

Here are some of the graphs that you can use to get more information about the execution of your account. So you have the response time graph, the throughput graph, the user event and workflow graph, and also the histogram. So let's try to look at each of those graphs in detail.

Let's started with the response time graph. The response time graph can be used to look at the general health of your NetSuite account. Now, for this one, you can see that there are three information that we can get from this graph. We can get the client, the network, and the server performance.

Now, by looking at this graph, you can see how your account is performing over a specific period in time. And you can see that performance either from the client side, the network, or the server, and why is that important for us. Now, if you're looking at the client side, you can see here that this can be a potential problem either with the customer's computer, or the user's computer, or this can be a problem with a script that is running on the client side.

So if you have a very high response time on your client side, that means that it's time to see if your users are using like an older or slower computer, or if there are any customizations or scripts that are running on the client side that can potentially be causing the slowdown.

Now, aside from the client, we also have network data. And this would tell us, OK, is the slowdown being caused by network traffic? Now, if the network traffic is, of course, slowed at this particular point in time, maybe you can just recheck your network configuration.

So this is not really something that you can be optimizing as a SuiteScript developer. But if you are a NetSuite administrator, this is something that is worth looking at. Then also, you have server information. So here, you can see, OK, is there something wrong with any customizations that you've added on the server side?

Maybe you've added too many user event scripts. Or maybe you've added too many suite lists, or schedule scripts that are running in the background that is causing the slowdown. So by looking at the response time graph, you can pinpoint exactly where the problem is. Is the problem on the computer? Is the problem on the pipeline or on the network? Or is the problem happening on the server? So again, this is very important tool you can use to optimize your implementation.

Aside from the response time, we can also look at the throughput. Now, the throughput gives us information about how many instances that record was accessed, and how many users are using this over a particular point in time. So for example, we can see here that, OK, maybe I am getting a lot of users and access on this particular record at 9 o'clock on a Monday, or maybe I'm just getting a lot of access at this particular time.

Now, by looking at the number of records being accessed at that time, and also the number of users accessing that record at that time, you know when to schedule, say, your scheduled script.

So for example, if you're seeing that, OK, it looks like my users are accessing at this particular time, or the bulk of the operation is happening at this time, then you want to schedule your scheduled script outside of that time, so that you don't slow the system down. Again, this is just something that you can look at if you look at the throughput.

There are a lot of other things that you can use this data for, but mainly, you will be using this data to know how many records is being asked, or how popular that particular record is among your users. And if your users are-- if a lot of users are accessing this record, one other consideration that you might want to think about would be governance.

Do you think that your script can handle any amount of data that's being accessed at this particular point? So again, knowing the throughput would be very important for customization.

Now, the user event and workflow graph will tell us how our script is performing for this particular record. We can see here that, OK, is the problem causing the script to slow down because of a workflow, or is it slowing down because of a user event script that's deployed to a record.

Now, this is something that would affect us directly, because this is something that we have full control of as a SuiteScript developer. So if there's something wrong with our script, or if our script is practically slowing down at a particular record, then we can go into that script, and then look at how we can optimize that code better. So this is a graph that I believe is something that you would be using quite a bit.

So the response time histogram is something that you can use to verify the execution of your records. Now, this would give you the data on the response time based on the amount of record instances for the record. Now, this should be fairly flat, or fairly static, because you would always have the same response time for a certain number of records.

For example, if you have, like if you're accessing, say, nine records, the response time would be averaging from 100 to maybe 120 milliseconds. But if you see that, OK, for the same number of records, I've accessed nine records, and it's spiked to like 200 or 300, and then for the other records maybe I'm accessing like 12 records, and it gives me the same response time, that you can see that there's just an anomaly in my record. Maybe there's something that happened at that particular access time that caused the spike.

But if you constantly see that the response time is high-- for example, I'm accessing nine records and it's always one full second before the user gets the information, then you know that there's something wrong. But again, if there's just a spike and everything else is fairly flat, then you know that it's just an anomaly and something that you can easily ignore.

So here, you can find out where is it a problem with customization that you've added, or maybe it's just something that happened that one time. So those are the different tools that you can use for optimization. So please do check out the Application Performance Management tool.

The last tool that we're going to look at would be SuiteBundler. And this is an important tool if you want a package of customizations. Now, this is something that you would use if you have a sandbox account, and if you want to transfer your account from, say, production-- or from sandbox to production. Or if you are a partner and you want to sell your own customizations SuiteBundler would be, again, something that you would use quite a bit.

To create a bundle, what you need to do is that you need to go to customization SuiteBundler, and then create bundle. Now, creating a bundle is very easy, because we have a step-by-step process of guiding you through creating the bundle, starting with configuring your bundle, then next would be adding additional information on your bundle.

And then on this screen, as you can see here, it'll ask you to select the different customizations that you want to add to your bundle. And the customizations that you can add can vary. It could be like workflows that you've created, saved searches, custom forms, custom fields, custom records, custom lists, or any other customization that you think would be part of your project, or is necessary for your project to work.

And then on the last step of the process, you would be asked to confirm are these things the elements or the components that you want to add to your bundle? Now, one thing that is very neat with the SuiteBundler tool is that it will automatically pick up any customization that you forgot that is a dependency on the things that you did remember.

For example, if I selected a script record, it will automatically pick up the actual script file that is on the file cabinet. So you don't need to select the script file, because the system will do that for you. Or let's say I selected a workflow and that workflow has a workflow action script attached to that.

You don't even need to select that workflow action script, because that'll automatically be picked up by the system. So again, this is something that is very convenient to use, especially if you want to transfer information like customizations from one account to another.

Remember that the reason why you want to use the SuiteBundler tool is that whenever you perform a customization and you move that customization from one account to another, the IDs will change. So in this example, you can see that I have a list of custom records on my sandbox account. And let's say if I want to transfer my performance review record, and also my furniture record to the production account, you can see here that the ID changed, because, again, it will just be based on the IDs that are on your target account.

So in my sandbox account, since I have several customizations there, and I only have a few on production, then it'll just renumber that based on what's on production. And again, if you change the internal ID, that can potentially affect your script if you did not script properly.

So you need to also remember to accommodate for changes in the ID. But again, whenever you are transferring one account or one customization to another, just make sure, do remember to use the SuiteBundler tool.

Lastly here, let's try to look at where you can find help. And again, this is something that we've already seen. But let's just try to review that, because it's important for us to know where we can find help once we start developing our own scripts.

So for more information, you can always go to the Help Center. So we have been using the Help Center a lot throughout this course. So please go ahead and just go to the Help Center for more information. You can also go to SuiteAnswers.

And I think SuiteAnswers is a better reference for help when you start developing on your own, because it'll give you more information than compared to what you can see in the Help Center. Because in SuiteAnswers, you have additional articles that our support team has added.

We also have videos and webinars that we might have recorded regarding the topic that you're looking for. So please do consider going to SuiteAnswers. Also, we have other training courses. So if you are interested in learning more about how you can optimize or customize your NetSuite account.

For example, if you want to learn about SuiteFlow, which is something that you can use for point and click customization and automation. Or if you want to try out integration. So if you are creating a full-blown integration account, then please do check out our suite of courses. You can find more information about that at www.NetSuite.com/training.

We also have training webinars, so that is happening quite frequently, and you would see that on the login page. So we do have some notices that we post on the login page. But just in case you've missed any of the webinars, all webinars would be posted in SuiteAnswers. So again, another good reason for you to go to SuiteAnswers for more information.

Also, we have the user group. So if you haven't joined the user group just yet, I do recommend that you join that. So that's usergroup.NetSuite.com. That is a free group that you can join. Just go ahead and join, and then try to ask any questions there if you need help.

Or if you know the answer to some of the questions, also give back to the community by answering questions. And if you're really stuck and you don't know what you're going to do regarding something that you're developing, you can always contact technical support. So you can send an email to technical support. Or you can call them by phone, so that you can get more information.

Now, regarding implementation, if you need assistance on actual implementation in your account, you can contact the professional services or the NetSuite partner channel. So for that, just go ahead and contact your account managers, or your NetSuite account managers.

This concludes module 14, Important **Considerations.**