Implementing Data-Driven Testing Using Datapools

Mike Kelly

First published on the Rational Developer Network
http://www.rational.net

You've decided to implement data-driven testing because it's faster, more robust, easier to maintain, and/or cheaper than your current testing method. Now the only thing you're wondering is, *how* are you going to implement it? You've come to the right place. In this article I'll show you how to implement what I believe to be a very good data-driven model for test automation, using the Rational datapools and their associated methods. I'll address some of the key concepts that need to be understood and some of the pitfalls to watch out for, as well.

This article assumes that you're already familiar with the theory and definition of data-driven testing. Just to make sure we're on the same page, though, here's a simple definition: In the case of automated test scripting, data-driven testing is setting up a script to test an application, having that script make calls to a dataset (a datapool, database, or some other source), and using the data found there within the script. If you need more background information, I'd suggest you take a look at "Improving the Maintainability of Automated Test Suites" by Cem Kaner.

What's a Datapool?

A datapool is simply a two-dimensional dataset, where rows represent records and columns represent fields that supply values for the variables in a test script during playback. This is the built-in method Rational provides to enable testers to generate data-driven tests. Values contained in datapools are commonly used to process controls but can also be used to affect the flow of script execution, allow for dynamic test verification, and other purposes.

Datapools can contain any kind of information you want. In past projects, I've put the following into datapools:

- data to be entered into controls
- data used to navigate through applications
- data to build test scripts "on the fly"
- paths to verification data, verification points, and images
- parameters for functions
- names of scripts to execute
- names of other datapools
- comments on the datapool itself

The list of what can go into a datapool is truly endless. But keep in mind that putting things into datapools is time consuming and at some point will reach the point of diminishing returns. It's best to start small, and then as you gain experience expand your datapool functionality as the project demands.

What Kinds of Applications Can Be Tested with Datapools?

Good candidates for testing with datapools are as follows:

- Applications that have complex navigational paths. You can enter all of the variations into one datapool and then maintain just one script to test all paths through the application.

- Applications with varying results based on input. Again, one script can be used to populate the data; both the data entered and the verification data can be stored in one datapool (or set of datapools).
- Large applications with numerous iterations. The benefit of datapools will best be seen in these applications, no matter what their domain. As the look, feel, and navigation of the application change, your test data probably will not. Pooling the data and drawing on it in your scripts will help you make sure you're maintaining as few scripts as possible.

Basic Commands for Manipulating Datapools

It's important that you know the basics of the commands used to manipulate datapools. The SQABasic language contains five basic commands for datapool manipulation, as shown in Table 1. I'll be referring to these commands and their functions throughout the rest of this article. More information and syntax for the commands can be found in the SQABasic language reference included with the Rational software.

Command	Function
`SQADatapoolClose`	Close the specified datapool.
`SQADatapoolFetch`	Move the cursor for the datapool to the next row.
`SQADatapoolOpen`	Open the specified datapool.
`SQADatapoolRewind`	Reset the cursor for the specified datapool.
`SQADatapoolValue`	Retrieve the value of the specified datapool column.

Table 1: *The SQABasic commands for manipulating datapools*

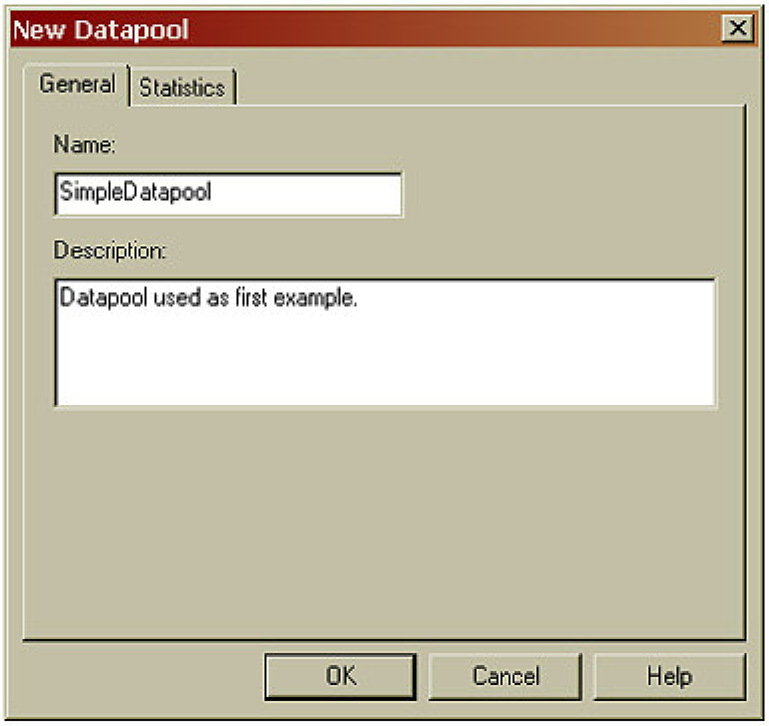
Using a Simple Datapool

All right, here we go. Let's make a simple datapool in TestManager to test a very simple application. After looking at the easiest case, we can then talk about applying the same concept to more complex systems. You can download the datapool if you'd like to take a look at it.

Create the Datapool

To create the datapool, do the following.

1. Open TestManager.
2. From the menu, choose Tools > Manage > Datapools > New.
3. In the New Datapool window, enter a name and a description for the datapool.



Name:

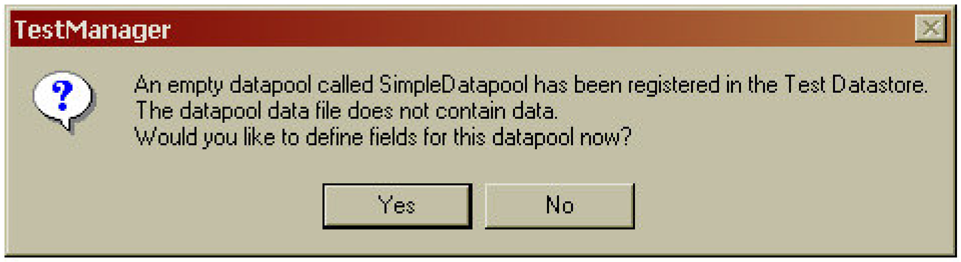
SimpleDatapool

Description:

Datapool used as first example.

Figure 1: *Make sure you enter a description of the datapool*

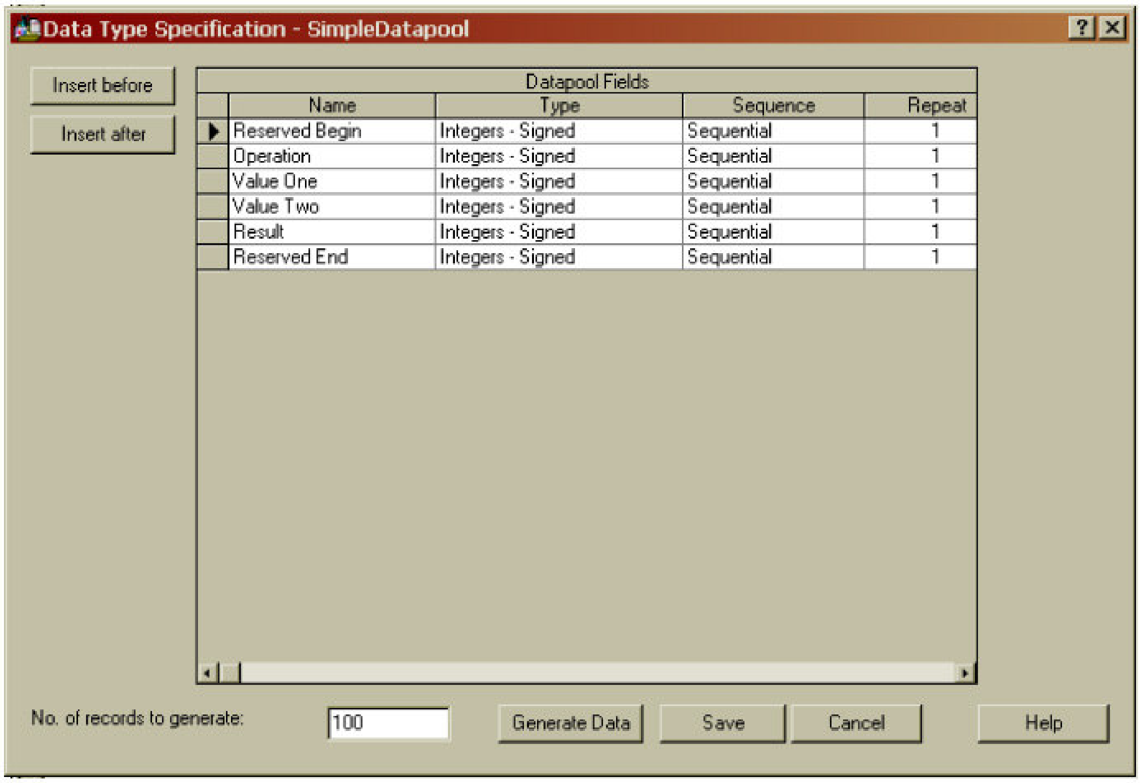
4. Click OK. The following pop-up should appear:



An empty datapool called SimpleDatapool has been registered in the Test Datastore.
The datapool data file does not contain data.
Would you like to define fields for this datapool now?

Figure 2: *Click Yes to define fields in your new datapool*

5. Click Yes. On the Data Type Specification screen, click the "Insert before" button six times and then enter the values shown below in the Name column of the six rows you've created:



Data Type Specification - SimpleDatapool

Datapool Fields

	Name	Type	Sequence	Repeat
▶	Reserved Begin	Integers - Signed	Sequential	1
	Operation	Integers - Signed	Sequential	1
	Value One	Integers - Signed	Sequential	1
	Value Two	Integers - Signed	Sequential	1
	Result	Integers - Signed	Sequential	1
	Reserved End	Integers - Signed	Sequential	1

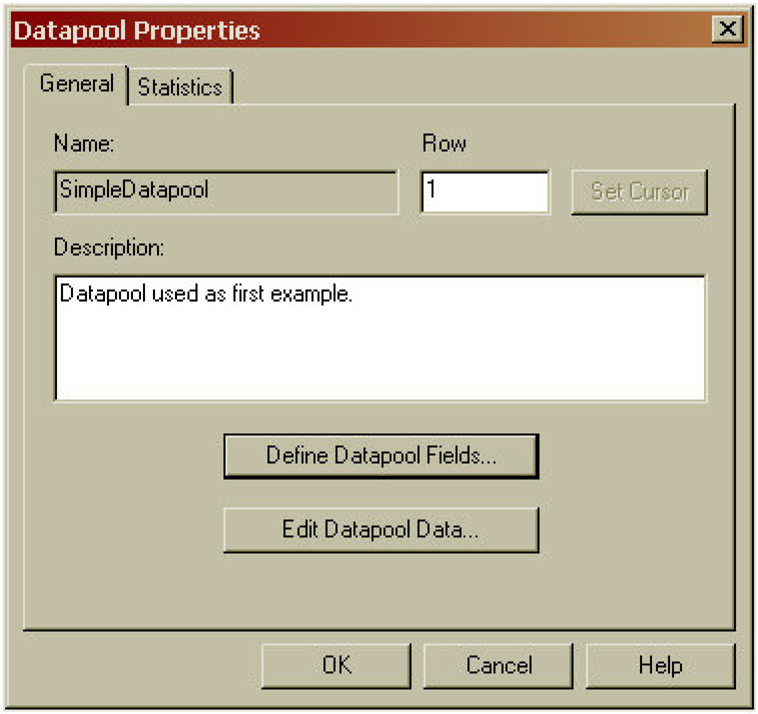
No. of records to generate: 100

Figure 3: *Inserting fields in your new datapool*

Regarding the entry in the first row, experience has taught me that in some complex datapool structures the values contained in the first and last columns of the datapool can be lost as a result of the framework of `SQADatapool` calls necessary to set up the structure. This may not happen in any implementations you set up, and it definitely won't happen in a datapool this simple. Nevertheless, I always reserve the first and last field, for two reasons. First, because debugging the various frameworks that had this problem was very costly in my experience, I figure that making this a standard practice is a good way to reduce potential headaches down the road. Second, these fields can be used to hold comments on the data contained in the row. This can be very helpful in large datasets.

I'm not going to go into detail on the other fields in the dataset or the Generate Data button. If you're unfamiliar with these features, when you have more time you can click the Help button on the screen and read the online help.

6. Click Save, then Close. You should be returned to the Datapool Properties screen.



Name: SimpleDatapool

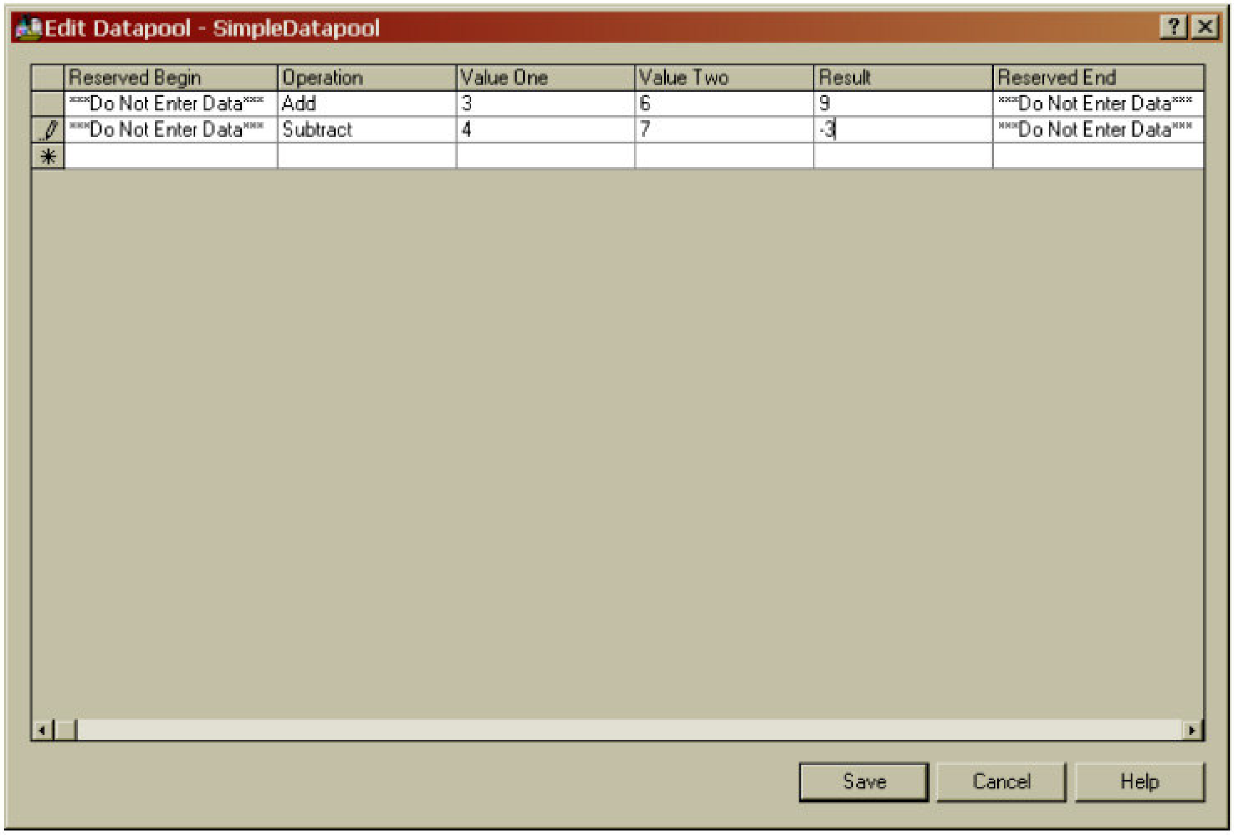
Row: 1

Description: Datapool used as first example.

Figure 4: *When you save the changes to your new datapool, you'll be returned to the initial screen.*

To go back to the Data Type Specification screen you were just at, you would click the Define Datapool Fields button. The values you just entered are the column names for the datapool. It's by those column names that you'll reference your data from within a script.

7. Click the Edit Datapool Data button. On the Edit Datapool screen, enter the values shown below in the first and second rows.

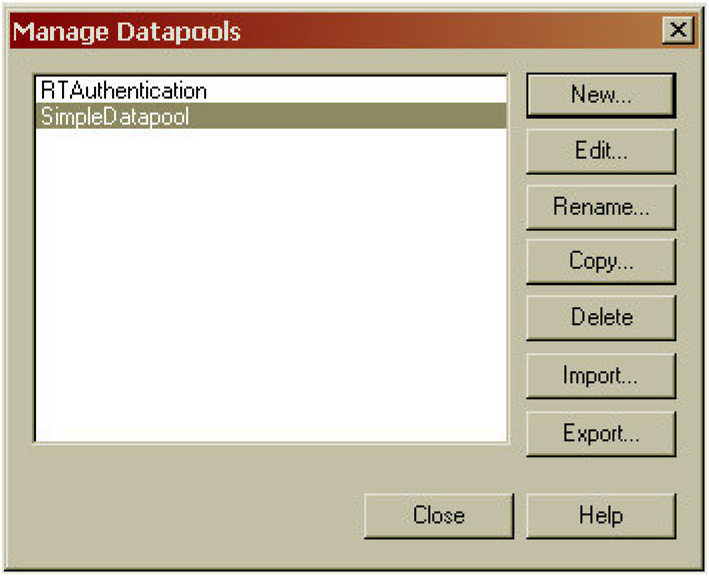


Edit Datapool - SimpleDatapool

Reserved Begin	Operation	Value One	Value Two	Result	Reserved End
Do Not Enter Data	Add	3	6	9	***Do Not Enter Data***
Do Not Enter Data	Subtract	4	7	-3	***Do Not Enter Data***

Figure 5: *Enter the values as shown above.*

8. Click Save, then Close. You should again be returned to the Datapool Properties screen.
9. Click OK, and you'll see that our datapool has been added.



Manage Datapools

RTAuthentication
SimpleDatapool

Figure 6: *Verify that your datapool has been added*

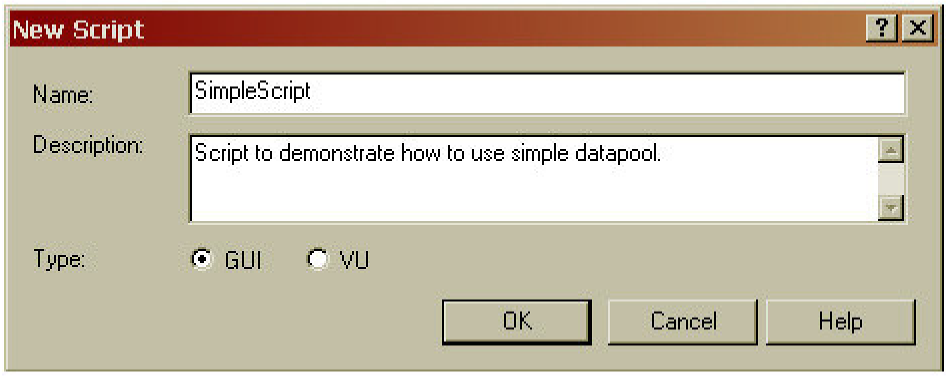
10. Click Close.

Create a Script to Use the Datapool

Now that we have our datapool, we can create a simple script to illustrate its use. We'll base the script on the Windows application Calculator so that you can follow along and create your own script. You can download the various versions of this script as we go along, if you'd like.

To create a script using our datapool, do the following:

1. Open Robot.
2. From the menu, choose File > New > Script.
3. In the New Script window, enter a name and a description for the script.



Name: SimpleScript

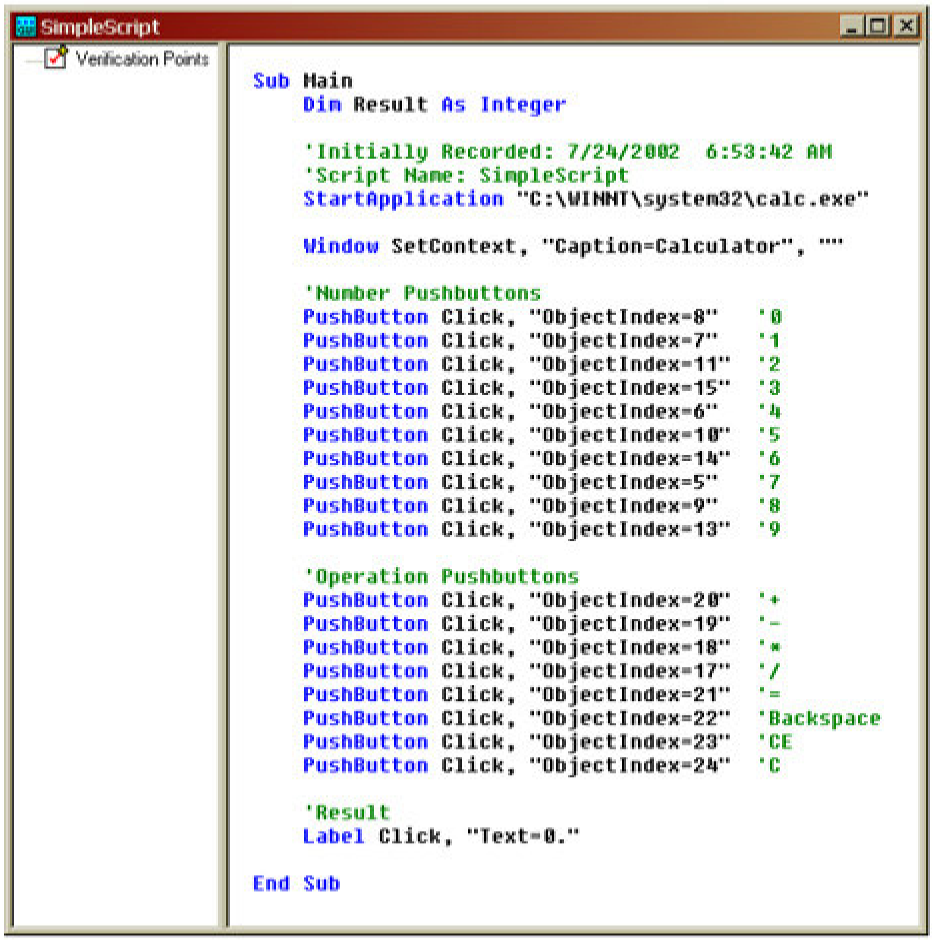
Description: Script to demonstrate how to use simple datapool.

Type: [✓ GUI] VU

Figure 7: *As usual, remember to add a description of your new script*

4. Click OK. A window titled SimpleScript should appear. From this window, enter a `StartApplication` command for the Windows Calculator.
5. Record as you click each button on the calculator as well as the results edit box.
6. Sort the buttons according to whether they're number buttons or operation buttons.

You should now have a script that looks similar to the one shown below. You can download this script; you'll have to compile it before it'll work.



```
Sub Main
    Dim Result As Integer

    'Initially Recorded: 7/24/2002  6:53:42 AM
    'Script Name: SimpleScript
    StartApplication "C:\WINNT\system32\calc.exe"

    Window SetContext, "Caption=Calculator", ""

    'Number Pushbuttons
    PushButton Click, "ObjectIndex=8"   '0
    PushButton Click, "ObjectIndex=7"   '1
    PushButton Click, "ObjectIndex=11"  '2
    PushButton Click, "ObjectIndex=15"  '3
    PushButton Click, "ObjectIndex=6"   '4
    PushButton Click, "ObjectIndex=10"  '5
    PushButton Click, "ObjectIndex=14"  '6
    PushButton Click, "ObjectIndex=5"   '7
    PushButton Click, "ObjectIndex=9"   '8
    PushButton Click, "ObjectIndex=13"  '9

    'Operation Pushbuttons
    PushButton Click, "ObjectIndex=20"  '+
    PushButton Click, "ObjectIndex=19"  '-
    PushButton Click, "ObjectIndex=18"  '*
    PushButton Click, "ObjectIndex=17"  '/
    PushButton Click, "ObjectIndex=21"  '=
    PushButton Click, "ObjectIndex=22"  'Backspace
    PushButton Click, "ObjectIndex=23"  'CE
    PushButton Click, "ObjectIndex=24"  'C

    'Result
    Label Click, "Text=0."

End Sub
```

Figure 8: *Make sure your script looks like this one*

Now we need something to test. For the sake of good testing, let's get some requirements. How about these:

- Verify the functionality of the Add button.
- Verify the functionality of the Subtract button.

Now that we have our challenging requirements, we can get down to work. Ideally we have everything we need to verify all of the requirements. Our datapool contains all of our test data, and we have one script, which can be used to test all (in this case both) requirements.

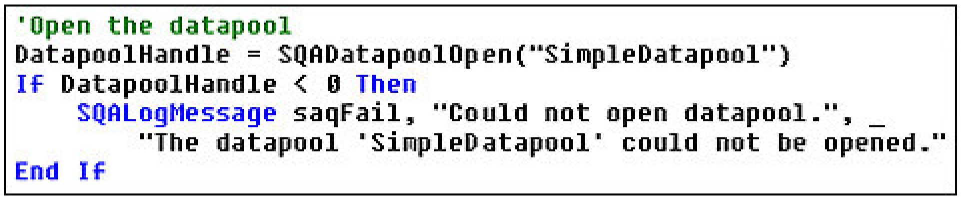
Add the Datapool to the Simple Script

To add our datapool to the simple script, do the following:

1. Open SimpleScript.
2. Include the header file `SQAUTIL.SBH`. This allows you to manipulate the datapool.

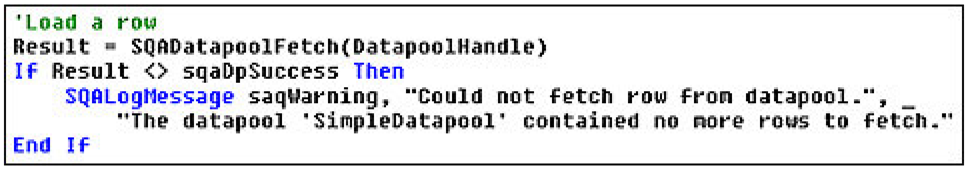
```
'$Include "SQAUTIL.SBH"

Sub Main
```

3. Add a call to `SQADatapoolOpen` at the start of the script. This will create a cursor for the datapool, allowing you to access the data. Always add error checking to allow for easier debugging.



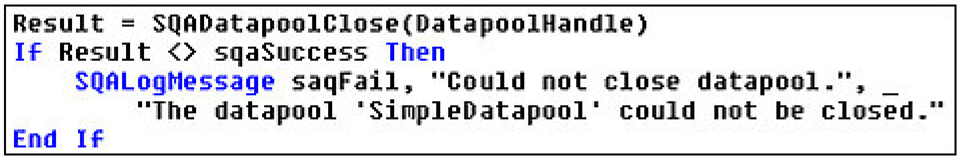
```
'Open the datapool
DatapoolHandle = SQADatapoolOpen("SimpleDatapool")
If DatapoolHandle < 0 Then
    SQALogMessage saqFail, "Could not open datapool.", _
        "The datapool 'SimpleDatapool' could not be opened."
End If
```

4. Add a call to `SQADatapoolFetch` after your `SQADatapoolOpen`. This will load the first row of the datapool into the cursor. The argument to this function is the datapool handle that was returned from the `SQADatapoolOpen`.



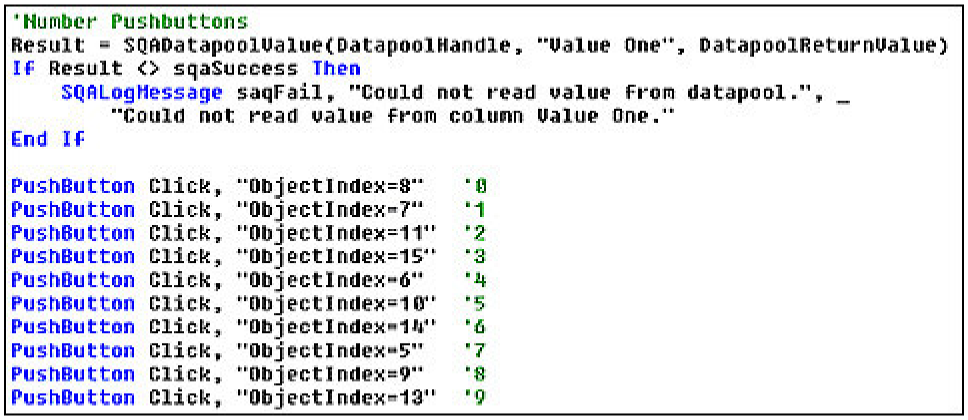
```
'Load a row
Result = SQADatapoolFetch(DatapoolHandle)
If Result <> sqaDpSuccess Then
    SQALogMessage saqWarning, "Could not fetch row from datapool.", _
        "The datapool 'SimpleDatapool' contained no more rows to fetch."
End If
```

5. Add a call to `SQADatapoolClose` at the end of the script. This will close the cursor after you're done processing the datapool.



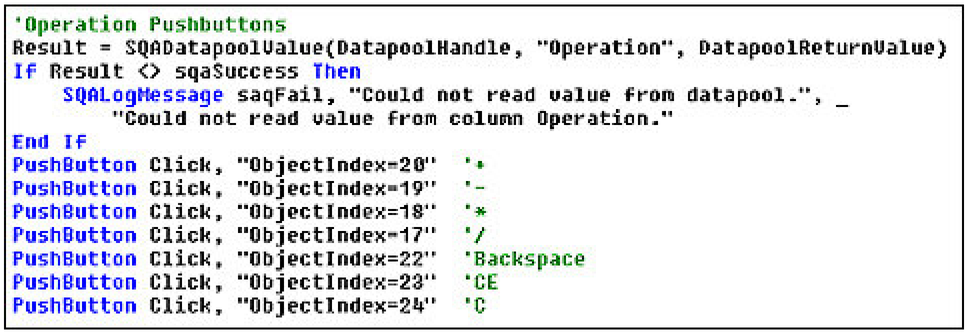
```
Result = SQADatapoolClose(DatapoolHandle)
If Result <> sqaSuccess Then
    SQALogMessage saqFail, "Could not close datapool.", _
        "The datapool 'SimpleDatapool' could not be closed."
End If
```

6. Add a call to `SQADatapoolValue` right before the number buttons to see which number to click. Pass the column name "Value One." This is one of the names we put in SimpleDatapool.



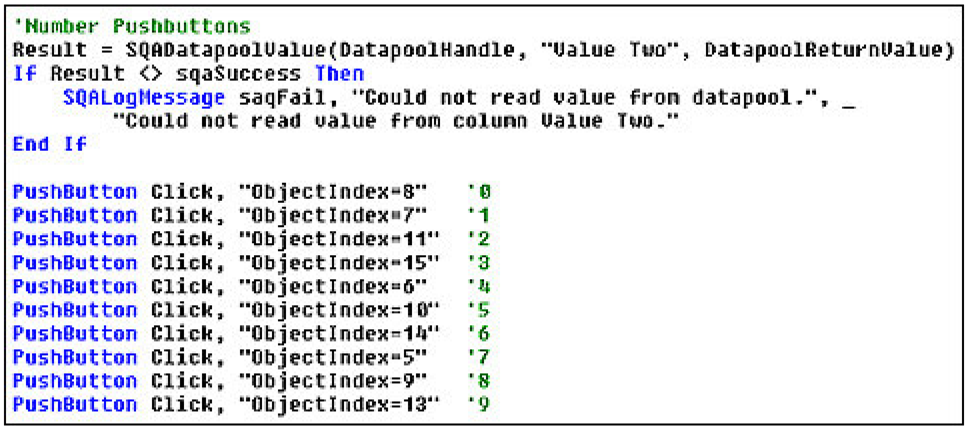
```
'Number Pushbuttons
Result = SQADatapoolValue(DatapoolHandle, "Value One", DatapoolReturnValue)
If Result <> sqaSuccess Then
    SQALogMessage saqFail, "Could not read value from datapool.", _
        "Could not read value from column Value One."
End If

PushButton Click, "ObjectIndex=8"   '0
PushButton Click, "ObjectIndex=7"   '1
PushButton Click, "ObjectIndex=11"  '2
PushButton Click, "ObjectIndex=15"  '3
PushButton Click, "ObjectIndex=6"   '4
PushButton Click, "ObjectIndex=10"  '5
PushButton Click, "ObjectIndex=14"  '6
PushButton Click, "ObjectIndex=5"   '7
PushButton Click, "ObjectIndex=9"   '8
PushButton Click, "ObjectIndex=13"  '9
```

7. Add a call to `SQADatapoolValue` right before the operation buttons to see which operation to click. Pass the column name "Operation." This is one of the names we put in SimpleDatapool.



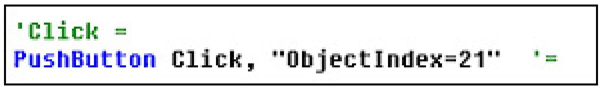
```
'Operation Pushbuttons
Result = SQADatapoolValue(DatapoolHandle, "Operation", DatapoolReturnValue)
If Result <> sqaSuccess Then
    SQALogMessage saqFail, "Could not read value from datapool.", _
        "Could not read value from column Operation."
End If
PushButton Click, "ObjectIndex=20"  '+
PushButton Click, "ObjectIndex=19"  '-
PushButton Click, "ObjectIndex=18"  '*
PushButton Click, "ObjectIndex=17"  '/
PushButton Click, "ObjectIndex=22"  'Backspace
PushButton Click, "ObjectIndex=23"  'CE
PushButton Click, "ObjectIndex=24"  'C
```

8. Copy the number buttons and paste them after the operation buttons (they should now be in your script twice). Add a call to `SQADatapoolValue` right before the number buttons to see which number to click. Pass the column name "Value Two." This is one of the names we put in SimpleDatapool.



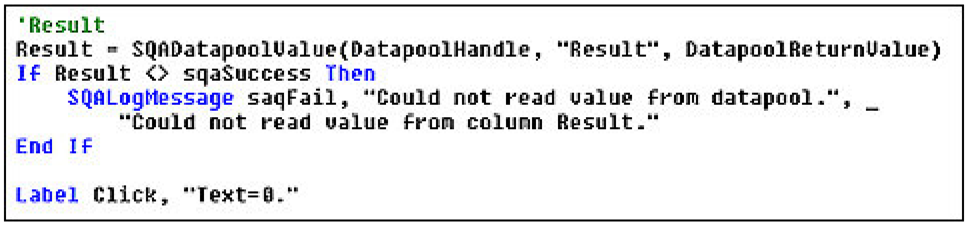
```
'Number Pushbuttons
Result = SQADatapoolValue(DatapoolHandle, "Value Two", DatapoolReturnValue)
If Result <> sqaSuccess Then
    SQALogMessage saqFail, "Could not read value from datapool.", _
        "Could not read value from column Value Two."
End If

PushButton Click, "ObjectIndex=8"   '0
PushButton Click, "ObjectIndex=7"   '1
PushButton Click, "ObjectIndex=11"  '2
PushButton Click, "ObjectIndex=15"  '3
PushButton Click, "ObjectIndex=6"   '4
PushButton Click, "ObjectIndex=10"  '5
PushButton Click, "ObjectIndex=14"  '6
PushButton Click, "ObjectIndex=5"   '7
PushButton Click, "ObjectIndex=9"   '8
PushButton Click, "ObjectIndex=13"  '9
```

9. Move the call to the "=" button after the second set of number buttons.



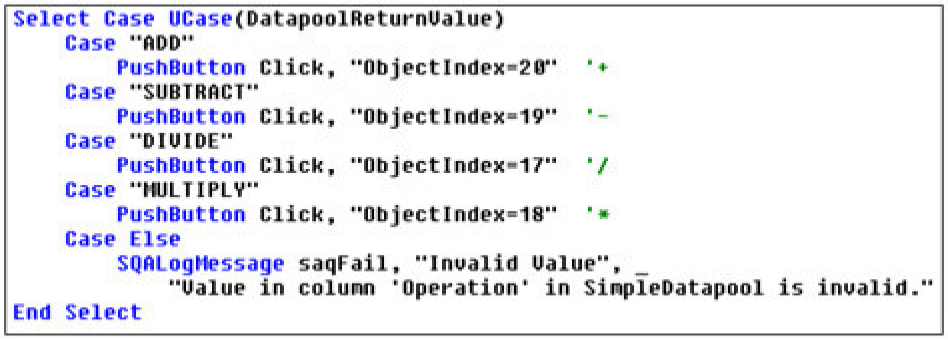
```
'Click =
PushButton Click, "ObjectIndex=21"  '=
```

10. Add a call to `SQADatapoolValue` right before the `Label Click` command for the result. Pass the column name "Result." This is one of the names we put in SimpleDatapool.



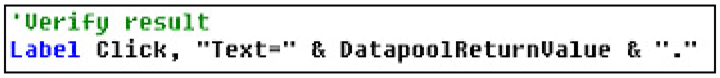
```
'Result
Result = SQADatapoolValue(DatapoolHandle, "Result", DatapoolReturnValue)
If Result <> sqaSuccess Then
    SQALogMessage saqFail, "Could not read value from datapool.", _
        "Could not read value from column Result."
End If

Label Click, "Text=0."
```

11. Add `Case` statements for the numbers and the operations. Below is an example of the operations.



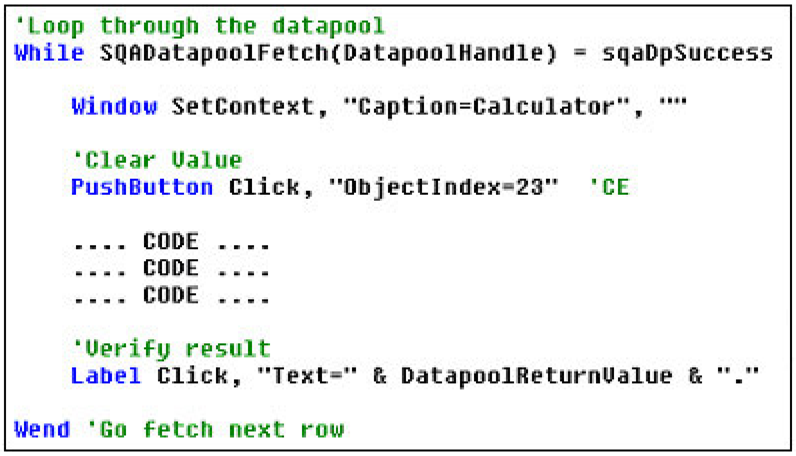
```
Select Case UCase(DatapoolReturnValue)
    Case "ADD"
        PushButton Click, "ObjectIndex=20"  '+
    Case "SUBTRACT"
        PushButton Click, "ObjectIndex=19"  '-
    Case "DIVIDE"
        PushButton Click, "ObjectIndex=17"  '/
    Case "MULTIPLY"
        PushButton Click, "ObjectIndex=18"  '*
    Case Else
        SQALogMessage saqFail, "Invalid Value", _
            "Value in column 'Operation' in SimpleDatapool is invalid."
End Select
```

12. The verification gets a little tricky. It can be done any number of ways, but for the purpose of our example, we'll let a click on the label with the desired result act as our verification. To do that we need to change the "Text" value of the label in our code by making it dynamic.



```
'Verify result
Label Click, "Text=" & DatapoolReturnValue & "."
```

13. If we were to execute the script at this point, it would process the first row of the datapool. Now we need to make the script truly dynamic and set it up to process multiple rows. We'll do this by looping until the datapool is empty. We'll also need to add a command to clear the calculator between tests. We add a `While` loop and use the `SQADatapoolFetch` command as the control for it.



```
'Loop through the datapool
While SQADatapoolFetch(DatapoolHandle) = sqaDpSuccess

    Window SetContext, "Caption=Calculator", ""

    'Clear Value
    PushButton Click, "ObjectIndex=23"  'CE

    .... CODE ....
    .... CODE ....
    .... CODE ....

    'Verify result
    Label Click, "Text=" & DatapoolReturnValue & "."

Wend 'Go fetch next row
```

At this point, if we were to run our script it would loop through the datapool and check every test case. You can download this script; you'll have to compile it before it'll work. Play around with variations and see what passes and fails. Also keep in mind that the script we made is an example. If you were to really create scripts like this one you would want to modularize some of the code (the number buttons, for example) as well as use wrapped functions instead of using the direct SQABasic commands.

Advanced Datapool Concepts

Now that you've successfully created and used a simple datapool, let's look at some more advanced implementations of datapools. As you start to use datapools for testing larger applications with larger datasets, there are a couple of key concepts you'll need to grasp.

First, there are three files for each datapool ? one file for each dimension (row, column) and one for meta-information.

- The data (row information) for the datapool is stored in a file with a `.csv` extension. You should already be familiar with this type of file as it's simply a comma-delimited list that can be edited with TestManager, Excel, or some other spreadsheet editor. I mention Excel because it can be a useful way of distributing blank spreadsheets to be populated with data by users who don't have TestManager. I've found in the past that this is very useful, because on large datasets data entry can sometimes be a bottleneck.

- The column names for the datapool are stored in a file with a `.spc` extension. The `.spc` file is formatted specifically for use with TestManager, and editing it outside of TestManager can easily corrupt the file.

- The meta-information for the datapool is stored in XML format with a `.rtxml` extension. The `.rtxml` file is formatted specifically for use with TestManager, and editing it outside of TestManager can easily corrupt the file.

All of the datapool files are stored in the TestDatastore under the following path:

`C:\\TestDatastore\DefaultTestScriptDatastore\TMS_Datapools`

You'll need to know where your datapools are on the physical disk if you want to edit them outside of TestManager.

Datapools can be nested to form datapool hierarchies. You'll want to customize the hierarchy structure to fit the structure of the application you're testing, not the test cases that will be executed. This means that you can start developing the datapool hierarchy at the same time the developers develop the application and the test engineers develop the test cases. Early data can later be put into the datapools contained in the hierarchy, and the bulk of your finished test cases can be executed with little or no scripting.

Let's take a look at a potential datapool hierarchy we could develop for the Calculator application.

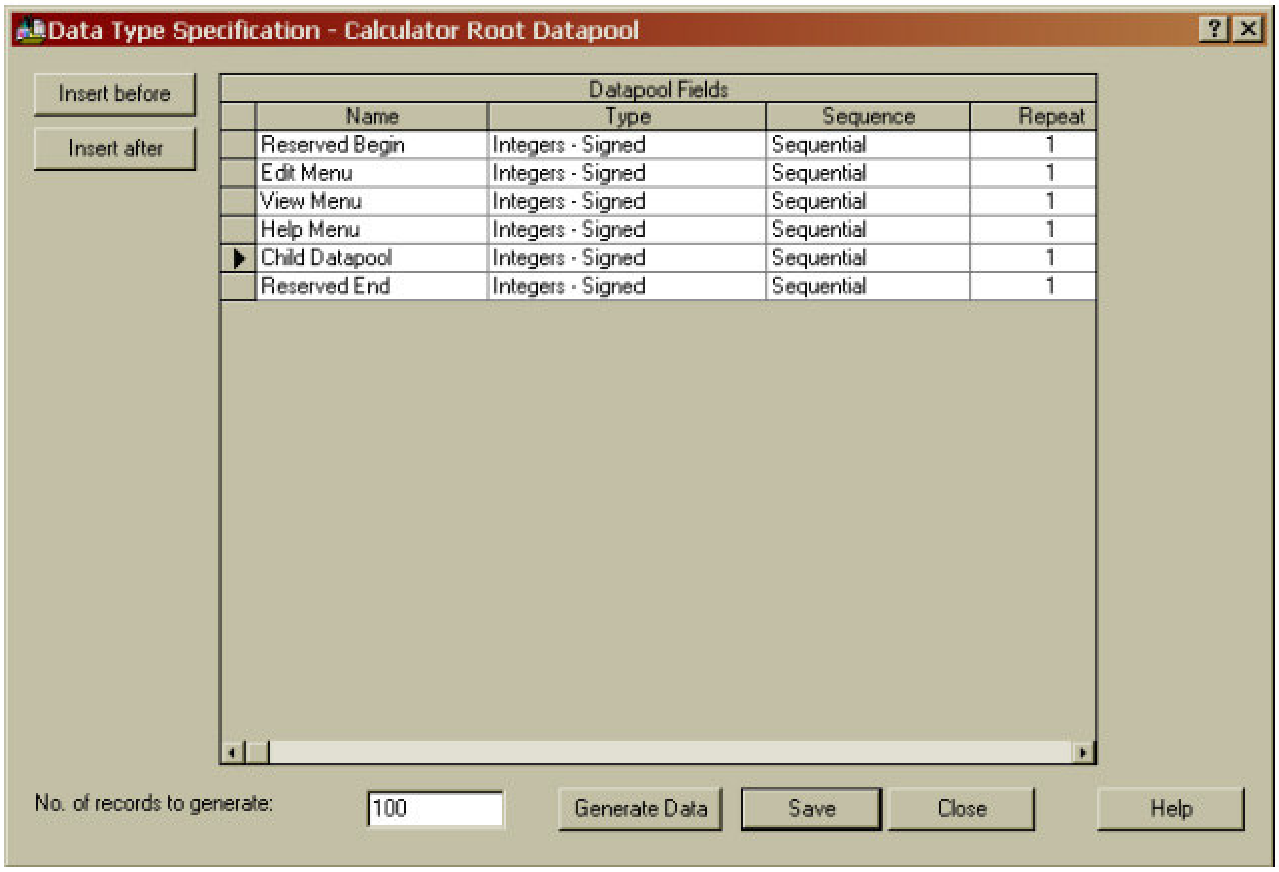
Create a Datapool Hierarchy

To create a datapool hierarchy, do the following:

1. Look at the application you'll be testing. For the example, we'll look at the Windows Calculator program.

2. Decide what the primary datapool (the root) will be and what it will contain.

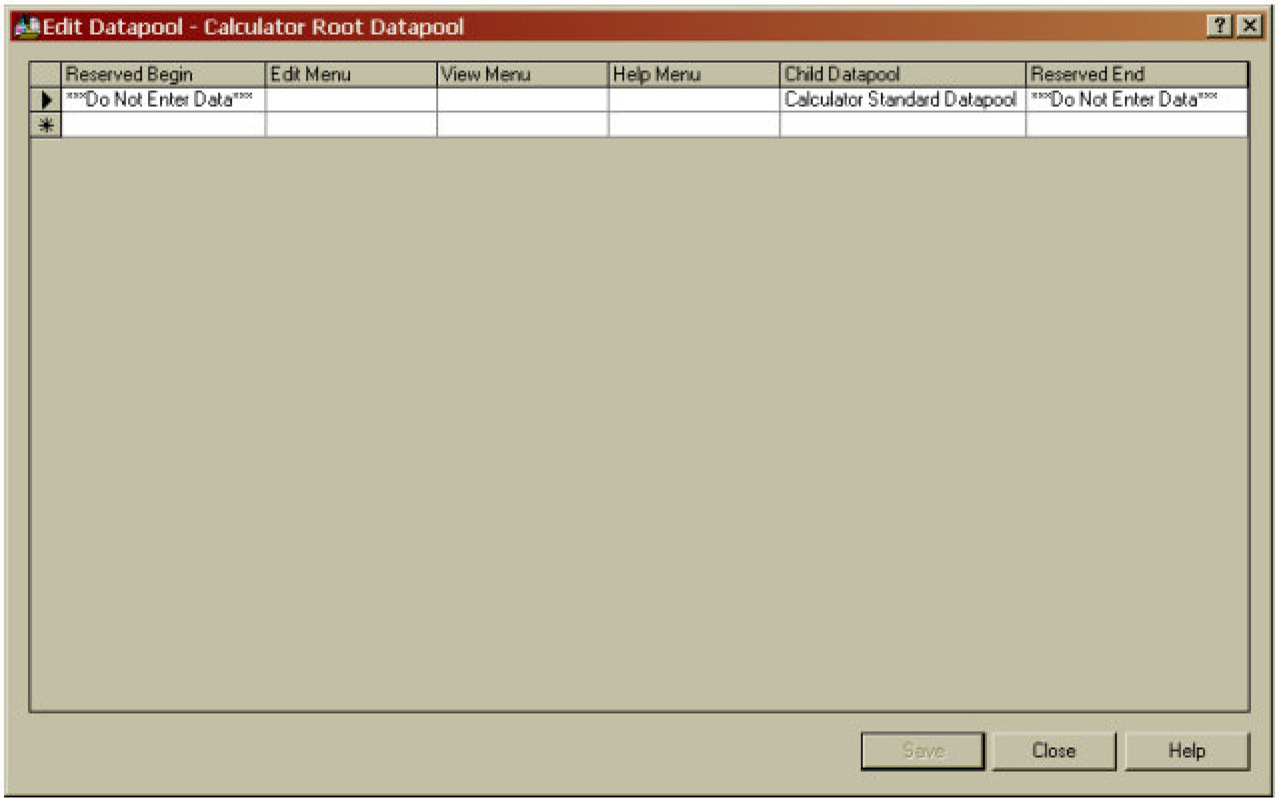
For the Calculator application, the primary datapool should contain columns for the menu commands and a pointer column for the child datapool (either the Standard or the Scientific view). With that information, our root datapool (named Calculator Root Datapool) would look like this:


Data Type Specification - Calculator Root Datapool
Insert before
Insert after
Datapool Fields
No. of records to generate:
Generate Data
Save
Close
Help

	Name	Type	Sequence	Repeat
	Reserved Begin	Integers - Signed	Sequential	1
	Edit Menu	Integers - Signed	Sequential	1
	View Menu	Integers - Signed	Sequential	1
	Help Menu	Integers - Signed	Sequential	1
▶	Child Datapool	Integers - Signed	Sequential	1
	Reserved End	Integers - Signed	Sequential	1

No. of records to generate: 100

Figure 9: *The calculator datapool*

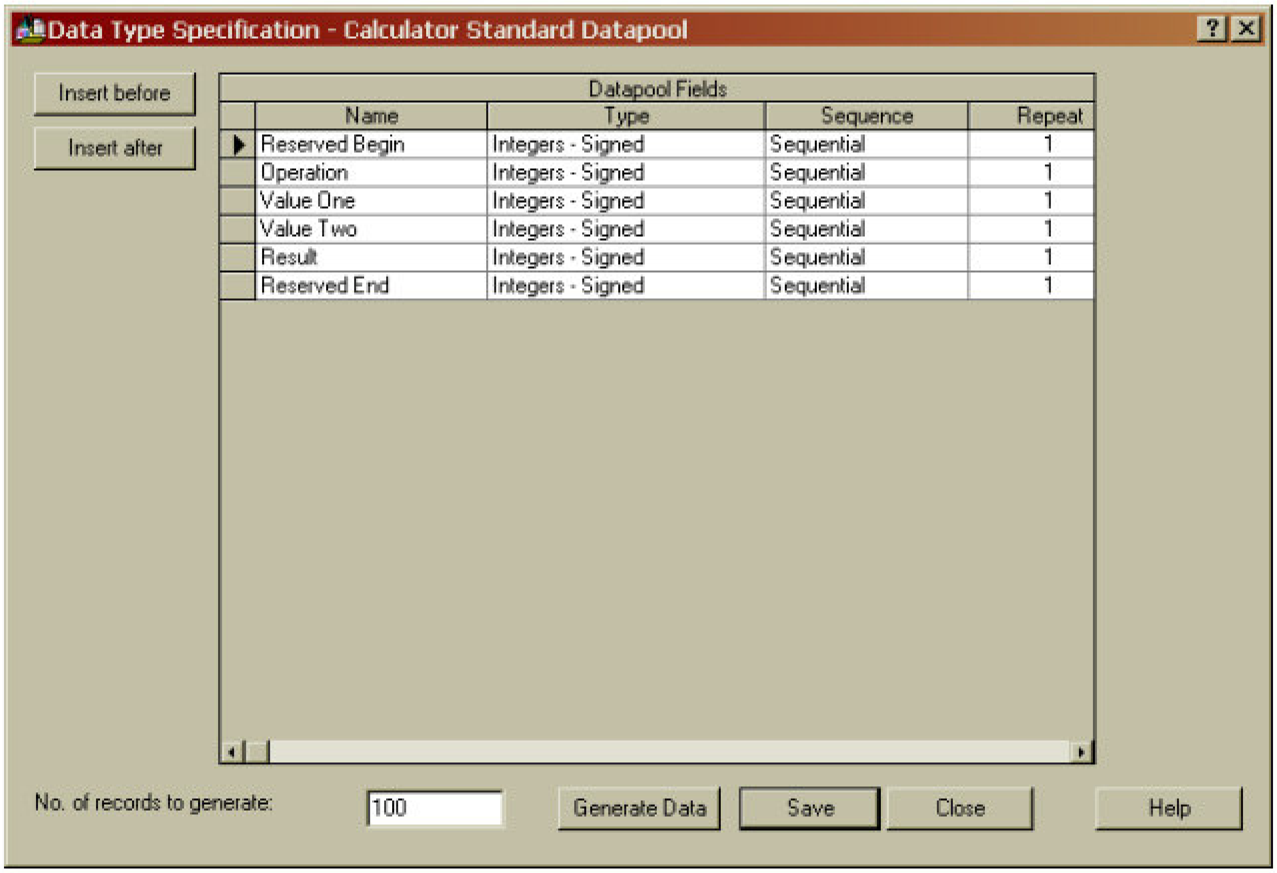


	Reserved Begin	Edit Menu	View Menu	Help Menu	Child Datapool	Reserved End
▶	***Do Not Enter Data***				Calculator Standard Datapool	***Do Not Enter Data***
✳						

Figure 10: *Begin mapping the root datapoool to the child datapools*

3. After the root datapool has been established, map out the child datapools it points to.

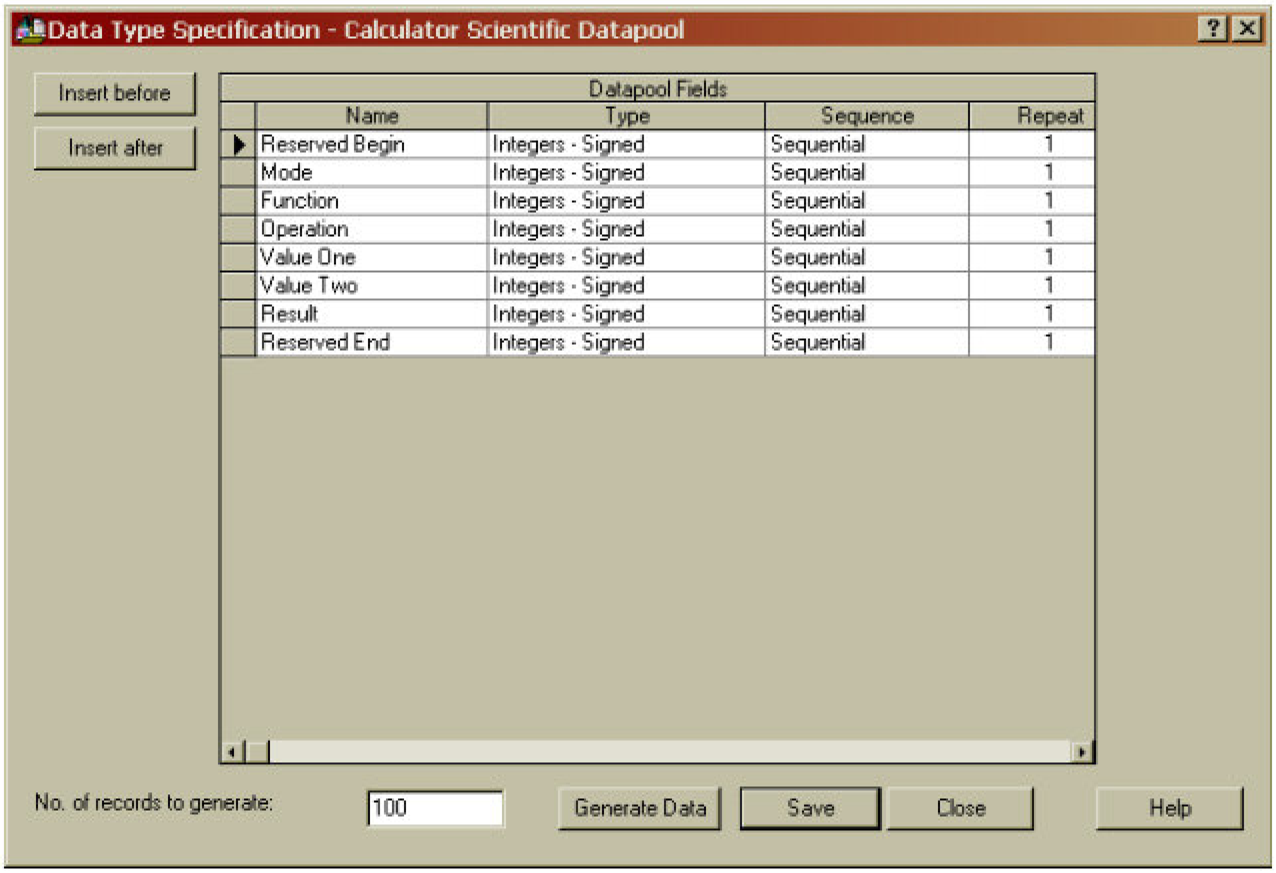
For the Calculator application, there are two child datapools: Calculator Standard Datapool and Calculator Scientific Datapool. They'll look like this:



	Name	Type	Sequence	Repeat
▶	Reserved Begin	Integers - Signed	Sequential	1
	Operation	Integers - Signed	Sequential	1
	Value One	Integers - Signed	Sequential	1
	Value Two	Integers - Signed	Sequential	1
	Result	Integers - Signed	Sequential	1
	Reserved End	Integers - Signed	Sequential	1

No. of records to generate: 100

Figure 11: *The standard calculator datapool*



Datapool Fields

	Name	Type	Sequence	Repeat
▶	Reserved Begin	Integers - Signed	Sequential	1
	Mode	Integers - Signed	Sequential	1
	Function	Integers - Signed	Sequential	1
	Operation	Integers - Signed	Sequential	1
	Value One	Integers - Signed	Sequential	1
	Value Two	Integers - Signed	Sequential	1
	Result	Integers - Signed	Sequential	1
	Reserved End	Integers - Signed	Sequential	1

No. of records to generate: 100

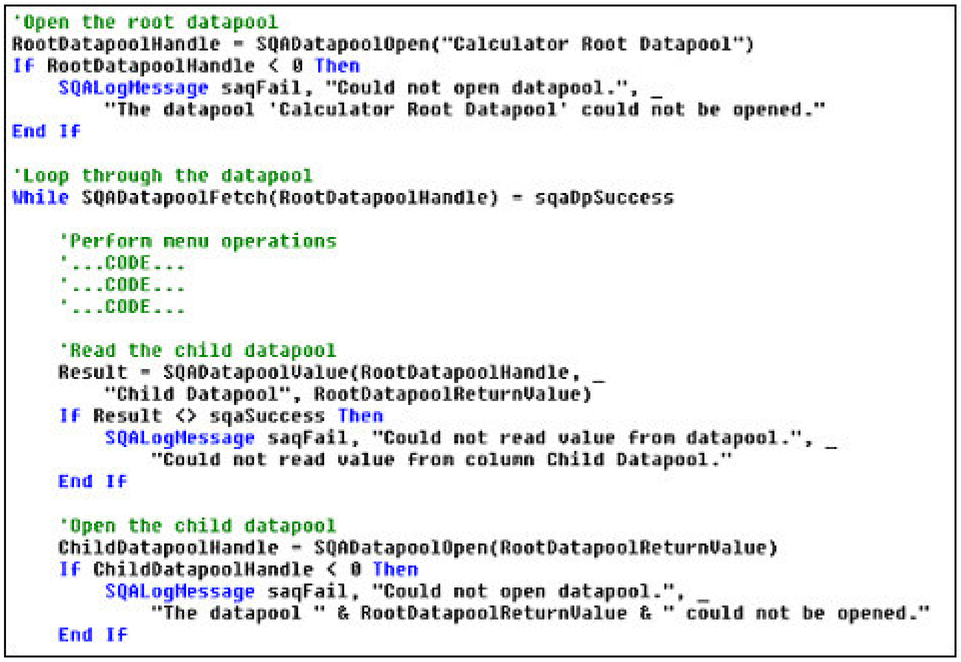
Figure 12: *The scientific calculator datapool*

Notice that the Calculator Standard Datapool is a copy of our SimpleDatapool that we created earlier. For the Calculator Scientific Datapool, all we added were columns for Mode and Function.

4. Go to those datapools and the sections of the application that they represent and repeat the process. When you've completely mapped out the application, you're done.

For the purpose of our Calculator example, we're done. We now have a very simple two-level datapool hierarchy.

When implementing these datapools in your scripts, you simply read the name of the nested datapool from the root (using `SQADatapoolValue`) and then pass that value to a `SQADatapoolOpen` that uses a different datapool handle, as shown below.



```
'Open the root datapool
RootDatapoolHandle = SQADatapoolOpen("Calculator Root Datapool")
If RootDatapoolHandle < 0 Then
    SQALogMessage sqaFail, "Could not open datapool.", _
        "The datapool 'Calculator Root Datapool' could not be opened."
End If

'Loop through the datapool
While SQADatapoolFetch(RootDatapoolHandle) = sqaDpSuccess

    'Perform menu operations
    '...CODE...
    '...CODE...
    '...CODE...

    'Read the child datapool
    Result = SQADatapoolValue(RootDatapoolHandle, _
        "Child Datapool", RootDatapoolReturnValue)
    If Result <> sqaSuccess Then
        SQALogMessage sqaFail, "Could not read value from datapool.", _
            "Could not read value from column Child Datapool."
    End If

    'Open the child datapool
    ChildDatapoolHandle = SQADatapoolOpen(RootDatapoolReturnValue)
    If ChildDatapoolHandle < 0 Then
        SQALogMessage sqaFail, "Could not open datapool.", _
            "The datapool " & RootDatapoolReturnValue & " could not be opened."
    End If
```

You can download this sample script; you'll have to compile it before it'll work.

Once you have this structure in place, you can create as many datapools as you need. For instance, if you had five test plans that called for testing the Scientific view, and each test plan had ten test cases, you would simply make five Calculator Scientific Datapool datapools, each with ten rows.

Pooling Your Resources

Implementing data-driven testing using datapools can be as simple or as complex as you need it to be. As you play around with datapools, datapool hierarchies, and different test frameworks, take some time to document how easy or hard it is to design and maintain specific implementations. Even if you find an implementation that works well for you, continue to make small changes to it with each new project you apply it to. Your automation framework should grow to become as robust as the applications you use it to test. As with all software development, you should always be trying to build a more robust and lower-maintenance system.

You're sure to save maintenance time when you implement a data-driven technique, but there are lots of different implementation methods to choose from. This article has described only one. I encourage you to look for more ideas on data-driven testing on the Rational Developer Network$^{(sm)}$. And if you have questions, feel free to email me.

References

Cem Kaner, "Improving the Maintainability of Automated Test Suites" (paper presented at Quality Week 97).

About the Authors

Mike Kelly is currently a programmer analyst for the Northeastern Center, Inc. He's had experience managing a software automation testing team and has been working with the Rational tools since 1999. His primary areas of interest are software development lifecycles, project management, and exploring new methods of software development. Mike can be reached by e-mail.