Test We Must!

In the lifecycle of the development process, there are few activities that I find *less* interesting than testing. Of all the skills that a developer has to have to write good software, sometimes I come up short when it comes time to find the bugs in a module of an application!

What is most frustrating is that 4D allows us to create superb programs very quickly and then we have to sit down and "waste" time looking for mistakes in our software!

But test we must for, without fail, if we release a segment of code that isn't thoroughly tested, that's the one that gives us the runtime error during the demo.

In addition to test plans and worksheets that we have developed to make testing easier, one simple addition to our process makes testing a little less onerous.

The method explained in this article accomplishes two important tasks—it helps us test our software more thoroughly and it helps reduce the time needed to test.

One of the major causes for failure to catch errors during the testing phase is lack of a sufficiently large amount of test data. In addition, when limited amounts of test data are created there is a good chance that the values in the data will not be sufficiently varied to test the quality of our defensive programming.

We can use the power of the programming language in 4th Dimension to quickly generate a large amount of test data that contains the wide variety of values required to test boundary checking.

How is It Done?

The first order of business is to create a large number of records for a specific file. The obvious solution is to use a simple For...End for loop and, for each pass of the loop, call Create record, assign the value, and then call Save record.

What is not quite so obvious is how to create records that contain the diversity of values that are required for testing. The key to this is to use a procedure that changes the values that are assigned to the records. This procedure must operate quickly, since we will want to be a to generate data in an interpreted database, and it must allow the developer to set minimum and maximum values for as many data types as possible.

The current version of 4th Dimension supports nine data types including alpha, text, real, integer, long integer, date, time, boolean, and picture fields. By using a procedure called “Make_Random”, we can easily generate random values for all field types.

“Make_Random” is passed two parameters. The first value is the minimum value, while the second value is the maximum value. The value passed back by the *Make_Random* function is a random integer value from within those limits.

```
Case of
  : (◊vonR)
     `Procedure: Make_Random
      `April 3, 1992
     `Douglas von Roeder
     `Desc: Returns a random number based on the values passed in.
     `Adapted from Foresight Technology's Online Help example

     `$1 is the lowest value in the range
     `$2 is the highest value in the range

     `Eg.:  Make_Random(1;10)
End case
C_INTEGER($0)
$0:=Random%($2-$1+1)+$1
```

Most of the data types make it easy to generate test data. When working with real, integer, long integer, date, and time fields all that is needed is to determine the minimum and maximum values and then pass those values to Make_Random. The integer returned from Make_Random can then be assigned to the field.

Time values are handled in a slightly different manner. 4th Dimension allows you to add and subtract time values but you must remember that 4th Dimension deals with time in seconds. To modify a time value by one minute, simply add (or subtract) 60*60 to that value.

The next step up the ladder of complexity is to create data for Boolean fields. To derive a Boolean value, pass a minimum value of 1 and a maximum value 2. The values that are returned can be tested to be equal to either a “1” or a “2”. Since the result of this test will be either **True** or **False**, we can capture a Boolean value.

String and text values can be generated as random values. Unfortunately, a randomly generated string of alpha characters has makes it difficult to verify. Instead of dealing with test values such as “kjjksdfajb” and “iohb wefB” it is easier to simply use a base value and then add a suffix that allows the tester to validate the entry.

The final data type, picture data, can be randomly assigned from a picture array. For example, if you load a picture array with 10 values you can pass 1 as the minimum value and 10 as the maximum value and use the random value to reference an element in the picture array.

A Sample Procedure

The procedure below shows how we can take advantage of Make_Random. The procedure “Crt_Inv_Recs” is shown below:

```
Case of
  : (◊vonR)
     `© 1993 SE Software, Inc.
     `Procedure: Crt_Inv_Recs
     `July 20, 1993
    `Douglas von Roeder

     `Creates data for testing purposes
End case
C_LONGINT($i;$Reps;$NumEmps;$NumCats)
$Reps:=250

 `1 Init the arrays that will be used as pick lists
ARRAY STRING(40;aEmpIDs;0)
ALL RECORDS([Employees])
SELECTION TO ARRAY([Employees]Entered_by_ID;aEmpIDs)
$NumEmps:=Size of array(aEmpIDs) `1 Used to create entries in [Employees]ID

LIST TO ARRAY("Inventory Categories";asInvCats)
$NumCats:=Size of array(asInvCats) `1 Used to create entries in
[Inventory]Category

 `ThermoInit($Reps;"Creating Inventory records...";"") `1 May as well watch
 `«_something_
For ($i;1;$Reps)
 CREATE RECORD([Inventory])
  [Inventory]ID:=Sequence number([Inventory])
  [Inventory]Buy:=Make_Random (25;35000)

   `1 Add a value from the array
  [Inventory]Category:=asInvCats{Make_Random (1;$NumCats)}
   [Inventory]SKU:=String(1234+$i)
  [Inventory]Comment_1:="Comment for SKU "+[Inventory]SKU

   `1 Add between 1 and 100 days to the [Inventory]Date_Entered
   [Inventory]Date_Entered:=!01/01/93!+Make_Random (1;100)
  [Inventory]Desc_1:="Desc for SKU "+[Inventory]SKU

   `1 Add a value from the array
 [Inventory]Entered_by_ID:=aEmpIDs{Make_Random (1;$NumEmps)}
  [Inventory]Qty_OH:=Make_Random (0;35)

    `1 True/false
```

```
   [Inventory]Returnable:=((Make_Random (1;2))=2)

    `1 Add 5 to 20 %
    [Inventory]Sell:=[Inventory]Buy*(1+(Make_Random (5;20)/100))

    `1 Add 1 to 6 %
    [Inventory]SRP:=[Inventory]Sell*(1+(Make_Random (1;6)/100))

    `1 Add between 1 second and 4 hours
    [Inventory]Time_Entered:=Current time+(Make_Random (1;60*60*4))

   [Inventory]Vendor_IDCode:=String(Make_Random (1;99))
   [Inventory]Weight:=Make_Random (2;15)
     `ThermoUpdate ($i)
  SAVE RECORD([Inventory])
End for
  `ThermoClose
UNLOAD RECORD([Inventory])
```

In this example, 4th Dimension will generate records to test a module that handles inventory records. In addition to working with different values, we had to accommodate different types of values including string, cost, weight, time, and date data. Thanks to a simple procedure such as “Make_Random” we were able to hundreds of test records in a manner of minutes.

This article has demonstrated a simple way to generate large volumes of test data for alphanumeric, boolean, date, and numeric fields. In addition, by changing just two values, you can control the upper and lower limits to ensure that boundary conditions are adequately tested.

Happy testing. Now, back to something a little more interesting!