9-1-00

| | | |
|---|---|---|
| **To:** | K. Crady | G. Ediss |
| | R. Groves | A. R. Kerr |
| | G. Lauria | S. -K. Pan |
| **cc:** | J. Webber | |
| **From:** | J. Effland | |
| **Date:** | 1 September 2000 | |
| **Subject:** | Software Design for Parameter Stepping | |

## Summary

SIS mixer performance is dependent on a large number of parameters, such as bias voltages, LO power levels and frequencies, magnet currents, *etc.* The measurement software must manage all of these parameters while providing the flexibility to allow any number of parameters to be stepped in any order, with start, stop, and step values easily defined and changed by the user.

The software described below provides this flexibility by using generic arrays of objects, where each object represents a particular instrument whose setting is defined by a parameter value. All the objects have common "set" and "read" functions, so the software can control all the instruments without knowing which instrument is being managed. Only the instrument driver, encapsulated in the instrument object, is unique to a particular instrument.

This memo describes how all these parameters are presented to the user and also presents the design of the parameter management classes.

This approach provides operational flexibility at the expense of software complexity and hence warrants this detailed software design. However, it is hoped that this design will produce software that is easier to maintain than simply writing more code each time a different sweep configuration is desired.

## Status

Coding of the main measurement dialog box is about 90% complete, with just a few bugs remaining to be corrected in the `Sweep Parameters` section. The classes `CParamMngr` and `CparamStepper` have been coded and tested. I am currently coding the class `CInstrCntl` and hope to have this new design operational in about 3 weeks.

## Main Measurement Dialog Box

Figure 1 shows the redesigned dialog box that is shown when the measurement program is run. The mixer of interest is highlighted in the `Mixers` section, and notes about that mixer are shown in the upper `Notes` grid. Measurements for that mixer and the notes for that measurement are listed in the central `Measurements` section.

The `Sweep Parameters` section at the bottom of the dialog box contains a grid that displays the parameter names and values for the particular measurement selected. The order that the parameters are listed in the grid determines how rapidly the parameter is changed. Those listed at the top of the grid change the most rapidly. Buttons to the right of the `Sweep Parameters` grid allows changing of parameter names, values, and sweep order. The `Sweep` column in this grid indicates whether the parameter is stepped or remains a constant value during the measurement. Constant values are identified by this column and also are located in a single column of the grid.

All the data in this dialog box is contained in the SQL server database running on my computer.

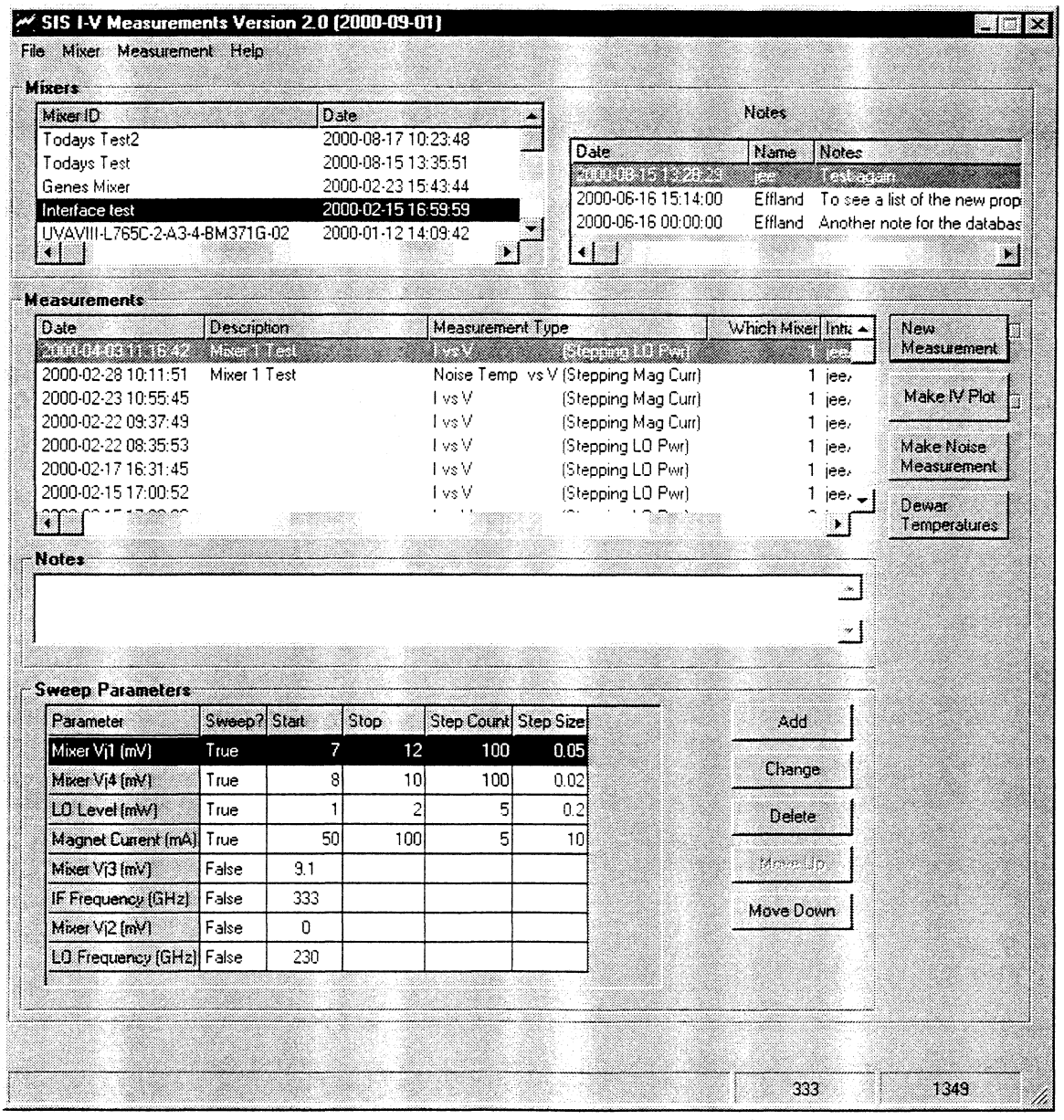


**Figure 1 : Main Measurement Interface Dialog**

# Swept Parameter Management

"Parameters" are defined here as characteristics that affect mixer performance, and include:

- Mixer 1 Bias Voltage
- Mixer 2 Bias Voltage
- Mixer 3 Bias voltage
- Mixer 4 Bias Voltage
- LO Frequency
- LO Power
- IF Frequency
- Magnet Current

The software design allows additional parameters, such as IF attenuation level, to be added by making database changes and adding an appropriate Instrument class. No changes are required to the control software.

The following sections provide design details on the parameter management classes. The class `CInstrCntl` controls each instrument. The instrument to be controlled and its value are obtained from the class `CParamMngr`. `CParamMngr` uses another class, `CParamStepper`, to control the individual parameter values.

## 1.1 CInstrCntl

Figure 2 shows how the class `CInstrCntl` performs the following funtions:

1. initializes a measurement, by obtaining the parameter information from the database, and building a generic array of parameter objects,
2. sets the relevant parameter to each instrument, and once all the parameters have been set, and
3. instructs the system to perform a measurement.

### 1.1.1 Initialization

First, this class queries the database to return all CW entries for this measurement, with the `SweepOrder` field sorted in ascending order. Given that recordset, it then instructs the calling program to set each of the instruments to the relevant CW value also returned from the database.

Next, it queries the database to return all sweep entries for this measurement, again with the `SweepOrder` field sorted in ascending order.

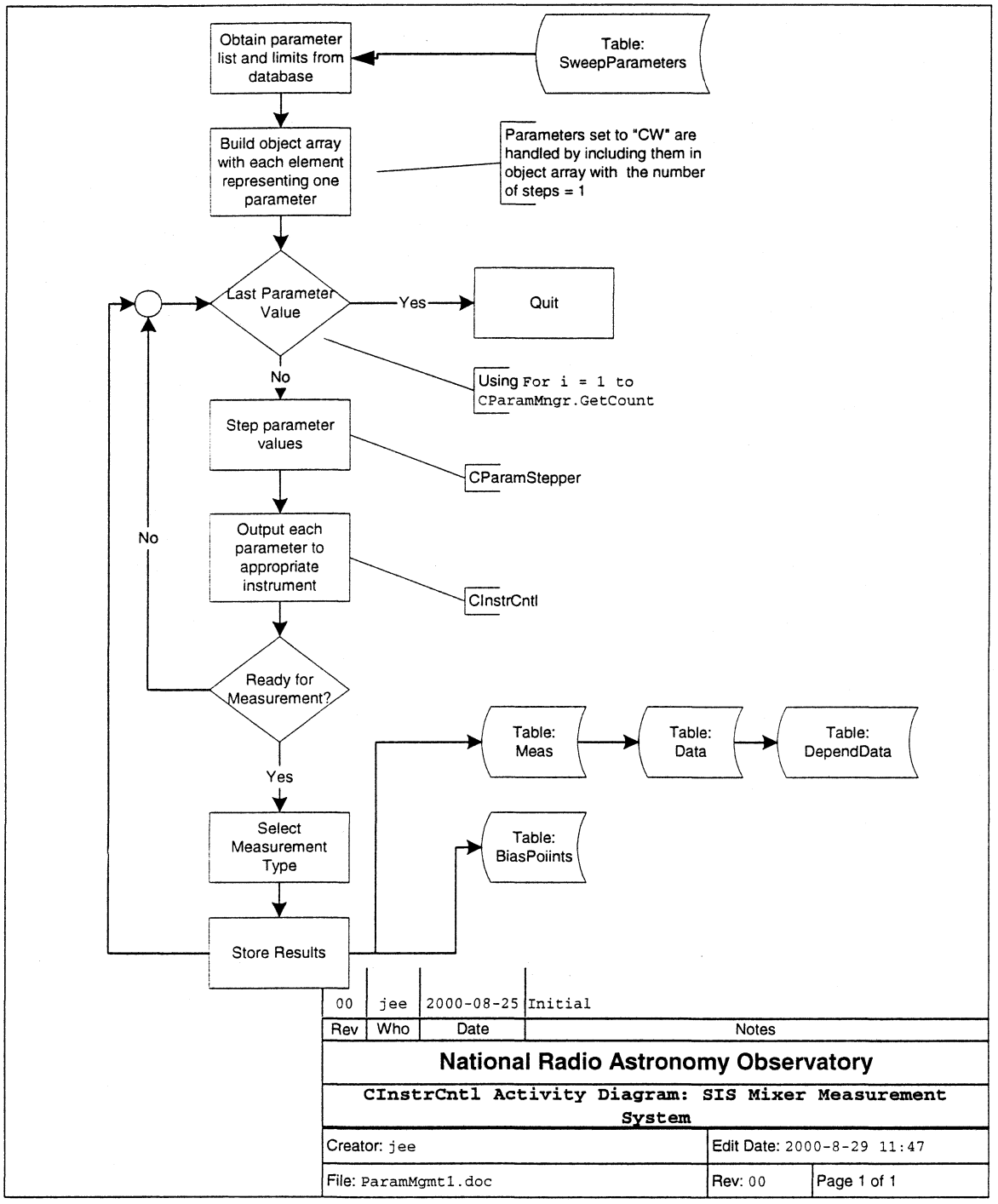


**Figure 2: Activity Diagram for `CInstrCntl`**

Figure 4 shows the sequence for setting and reading the instruments during a measurement. The object `oInstrCntl` makes a `bGenNext()` call to `CParamMngr`, which returns with an array of objects. Note the use of common function names, such as `bSet` and `bRead`, to facilitate using arrays of objects to make these calls generically. Some of these functions require more than one argument, and declaring the arguments as variant data types, which can handle either arrays or scalar values, readily accommodates that.

An example of generically setting and reading instruments follows. Error checking code is omitted for clarity:

```
Dim vInstruments as CInstrCntl                  ' array of instrument objects
Dim oParams as CParamMngr                       ' parameter manager object
Dim vData as variant                            ' array of variants to hold data

Do
        ' Get the next parameter value
        Do
                oParams.bGenNext(iInstrument, iState, vData)

                If iState = gParamStateChanged then
                        ' Set the instrument to this parameter value
                        vInstruments(iInstrument).bSet(vData
                End If

        Loop until iState = gParamStateTakeData


        ' Read the instruments
        For i = lBound(vInstruments) to uBound(vInstruments)
                vInstruments(i).bRead(vData
        Next I
Loop Until iState = gParamStateFinshed
```

With the present state of the hardware, some of the instruments cannot read back their parameters;

LO frequency
LO power
IF Frequency

and objects representing these instruments will store their command parameters (passed with `bSet`) and return the same command parameter when responding to the `bRead` function.

## 1.2 CParamMngr

The class `CparamMngr` is responsible for calculating the settings of all the instruments during a stepped measurement. Each parameter usually represents a unique instrument (such as Mixer 1 bias supply, Mixer LO power, etc.). `CparamMngr` builds a generic array of parameters (using repeated calls to `bInit`) to represent each instrument, and calls to `bGenNext` return which instrument to set next and the value to set it. The "State" parameter in `bGenNext` also identifies when all the instruments have been set so that it's time to make a measurement.

Key methods of this class are:

```
bInit(iType, dStart, dStop, lNumOfSteps)
```

where:

`iType` is the type of parameter (e.g., Bias Supply 1, LO power)
`dStart` – start value for the parameter
`dStop` – stop value for the parameter
`lNumOfSteps` – total steps parameter should make

```
bGenNext(iType, iState, vValue)
```

where:
`iType` – identifies the parameter,
`iState` - provides the state of this parameter (see below)

`vValue` – the new value(s) for the parameter.

The return value `iState` contains one of the following constants:

| | |
|---|---|
| `gParamStateSame` – | the parameter remains the same as the last call. This means the instrument is already set to this value. |
| `gParamStateChanged` - | the parameter value has changed from the last call |
| `gParamStateFinshed` - | the algorithm has stepped through all the states, so this signals completion of the stepping mode. |
| `gParamStateTakeData` - | All the parameters for this step have been set, so it's time to take data. |

The activities in `bGenNext` are illustrated in Figure 3. The method `bGenNext` uses an array of `CParamStepper` objects to manage incrementing values and determining when the "Terminal Count" has been set for each object. When "Terminal Count" is set for a `CParamStepper` object, then a lower priority parameter is sequenced.

The following code snippet from

`\\eagle\cv-cdl-sis\MeasSys\Software\BiasMeasV2.0\ParamStepper.xls`

is an example of using the `bGenNext` function to generate a table of values such as shown in Table 1. The code is shown below:

```
Dim oParams As CParamMngr
Dim lCtr As Long
Dim vParamValue As Variant
Dim iType As Integer
Dim iState As Integer


Set oParams = New CParamMngr

If Not oParams.bInit(1, 0, 5, 10) Then
    ' Error during init
    Call MsgBox(oParams.sGetError)
End If

If Not oParams.bInit(2, 10, 20, 2) Then
    ' Error during init
    Call MsgBox(oParams.sGetError)
End If

If Not oParams.bInit(3, 1, 2, 1) Then
    ' Error during init
    Call MsgBox(oParams.sGetError)
End If

lCtr = 1
Do
    If Not oParams.bGenNext(iType, iState, vParamValue) Then
        Call MsgBox(oParams.sGetError)
    End If

    If iState = gParamStateTakeData Then
        ' move to the next row after each "TakeData" command
        lCtr = lCtr + 1
    Else
        ActiveSheet.Cells(lCtr, iType).Select
        ' write the parameter value to the first column of this row,
        ' and the state to the second column
        ActiveSheet.Cells(lCtr, iType).Value = vParamValue
        ActiveSheet.Cells(lCtr, iType + oParams.iGetCount).Value = iState
    End If
Loop Until iState = gParamStateFinshed
```

```
Set oParams = Nothing
```

**Table 1 : Sample Parameter Value Stepping Using CParamMngr.bGenNext**

| Values | | | States | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 1 | 2 | 3 |
| 0 | 10 | 1 | 1 | 1 | 1 |
| 0.5 | 10 | 1 | 1 | 0 | 0 |
| 1 | 10 | 1 | 1 | 0 | 0 |
| 1.5 | 10 | 1 | 1 | 0 | 0 |
| 2 | 10 | 1 | 1 | 0 | 0 |
| 2.5 | 10 | 1 | 1 | 0 | 0 |
| 3 | 10 | 1 | 1 | 0 | 0 |
| 3.5 | 10 | 1 | 1 | 0 | 0 |
| 4 | 10 | 1 | 1 | 0 | 0 |
| 4.5 | 10 | 1 | 1 | 0 | 0 |
| 5 | 10 | 1 | 1 | 0 | 0 |
| 0 | 15 | 1 | 1 | 1 | 0 |
| 0.5 | 15 | 1 | 1 | 0 | 0 |
| 1 | 15 | 1 | 1 | 0 | 0 |
| 1.5 | 15 | 1 | 1 | 0 | 0 |
| 2 | 15 | 1 | 1 | 0 | 0 |
| 2.5 | 15 | 1 | 1 | 0 | 0 |
| 3 | 15 | 1 | 1 | 0 | 0 |
| 3.5 | 15 | 1 | 1 | 0 | 0 |
| 4 | 15 | 1 | 1 | 0 | 0 |
| 4.5 | 15 | 1 | 1 | 0 | 0 |
| 5 | 15 | 1 | 1 | 0 | 0 |
| 0 | 20 | 1 | 1 | 1 | 0 |
| 0.5 | 20 | 1 | 1 | 0 | 0 |
| 1 | 20 | 1 | 1 | 0 | 0 |
| 1.5 | 20 | 1 | 1 | 0 | 0 |
| 2 | 20 | 1 | 1 | 0 | 0 |
| 2.5 | 20 | 1 | 1 | 0 | 0 |
| 3 | 20 | 1 | 1 | 0 | 0 |
| 3.5 | 20 | 1 | 1 | 0 | 0 |
| 4 | 20 | 1 | 1 | 0 | 0 |
| 4.5 | 20 | 1 | 1 | 0 | 0 |
| 5 | 20 | 1 | 1 | 0 | 0 |
| 0 | 10 | 2 | 1 | 1 | 1 |
| 0.5 | 10 | 2 | 1 | 0 | 0 |
| 1 | 10 | 2 | 1 | 0 | 0 |

## 1.3 CParamStepper

This class is responsible for calculating the next value for individual parameters. It also determines when the "Terminal Count" has been reached, which means that the particular parameter has counted up to its maximum value. `CparamStepper` manages individual parameters while `CParamMngr` manages the set of all relevant parameters. That is, `CParamMngr` manages an array of `CParamStepper` objects.

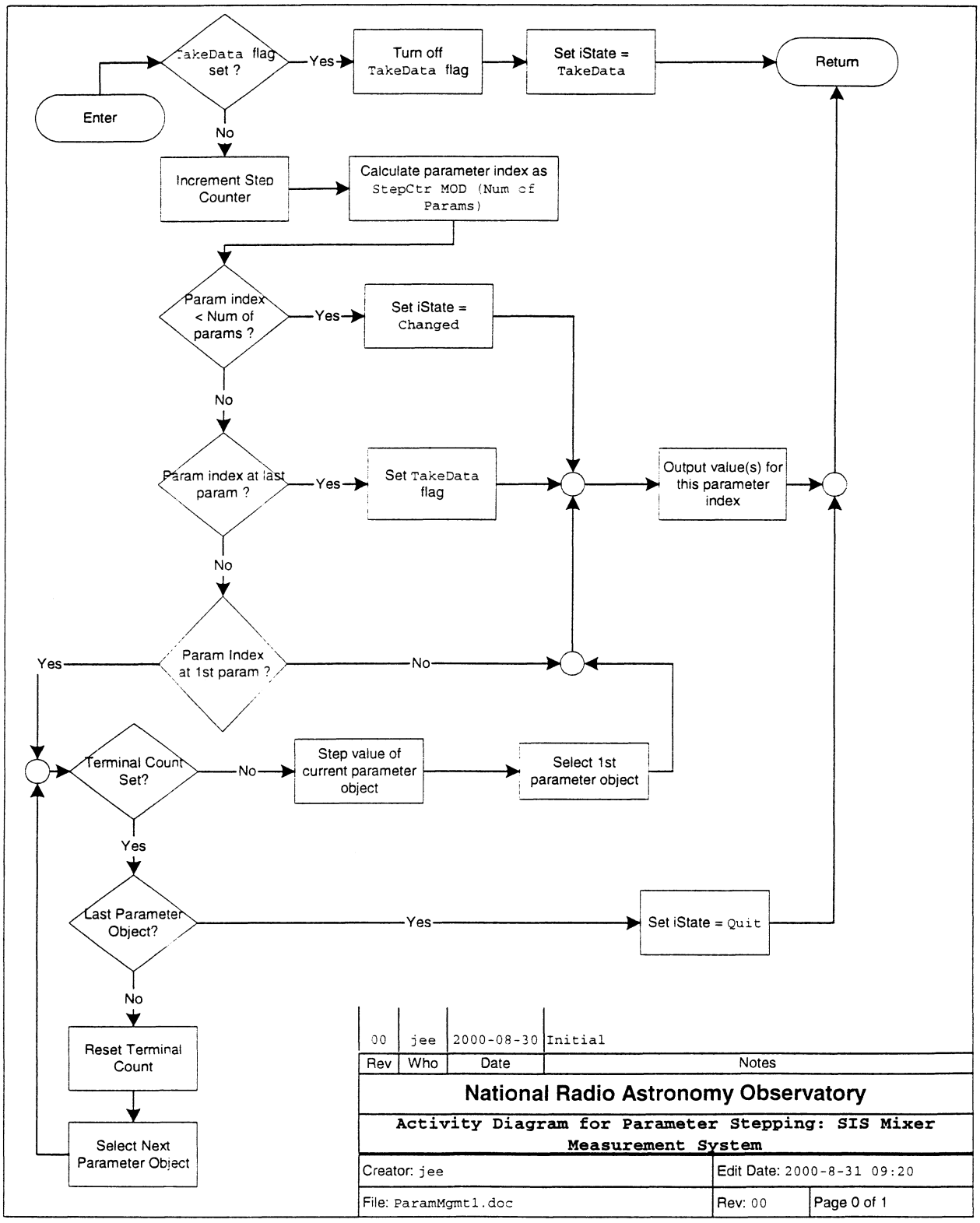


**Figure 3: Activity Diagram for Parameter Stepping (`bGenNext`)**

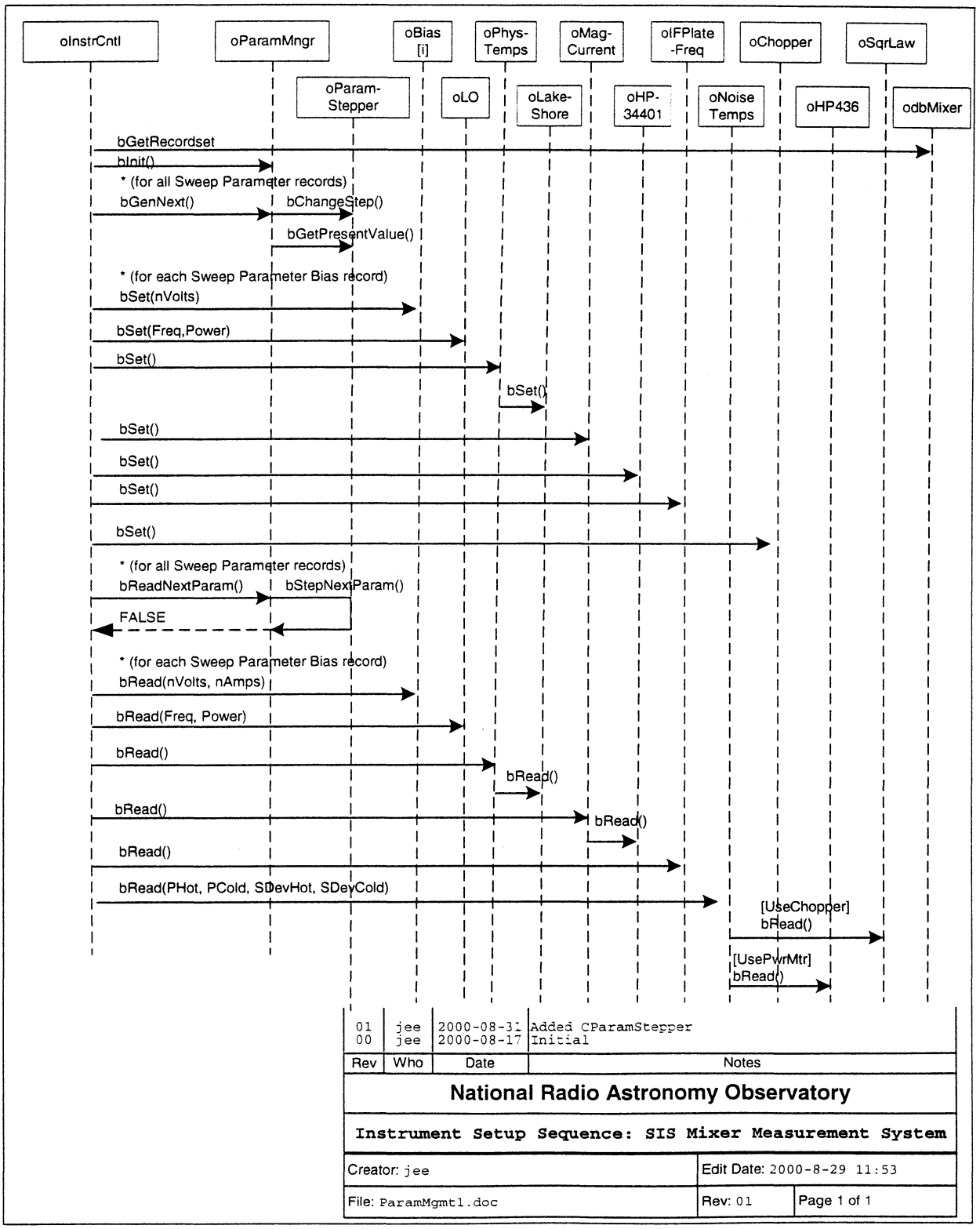


**Figure 4: Sweep Parameters Sequence Diagram**