3-3-00

Memorandum

To: K. Crady, G. Ediss, R. Groves, A. R. Kerr, S. -K. Pan, J. Pisano

cc: J. Webber

From: J. Effland

Date: 3 March 2000

Subject: Updates to SIS Mixer Data Acquisition Software to Include Noise Temperature Data

The next phase of the SIS mixer measurement system requires measuring and storing noise temperatures so that the software can ultimately determine an optimum operating point. This memo discusses the design of the noise temperature acquisition software.

At present, noise temperature is measured semi-automatically but only after the optimum mixer operating point is found. The results are stored in Excel spreadsheets. The motivation for this approach was to quickly enhance the entirely manual techniques used previously. Software that plots, in real-time strip chart form, Y-Factor as a function of time using the chopper wheel was also completed in January, but that only provides a convenient display for the operator to manually find the optimum operating point.

Current effort involves using the chopper wheel to rapidly measure and store in the database data that allows the calculation of noise temperature as a function of bias voltage, LO frequency, LO power, magnetic field, and IF frequency. Following this phase, the Origin graphing program will be interfaced to the measurement system for plotting subsets of the results on 3-D surface graphs. Finally, routines will be developed to search this multidimensional space and find the optimum operating point.

Attachment 1 is from the Software Design Document and describes this design approach. Please note that all of the routines and interconnecting logic shown in Figure 7 are already coded, except for `MeasureResults`. In fact, the process diagrammed in Figure 7 is now used to measure I-V curves. Even the routine `MeasureResults`, which is further defined in Figure 8, has the more-difficult modules already coded. The `CtrgSqrLaw` class is nearly complete: It now returns y-factors 8 times per second by averaging 1024 measurements. However, our software requirements state that absolute hot load and cold load power levels should be stored, so this class requires some minor revisions. Kirk Crady has just finished coding `CSpecAnal`.

Attachment 2 provides details on database refinements required during this stage. The present database schema has worked well for IV measurements, but requires some changes to optimally store noise temperatures and all the associated parameters.

Please forward any comments to me.

Attachment 1: Noise Measurement Software Flow Charts

Program flow for the main data acquisition loop is shown in Figure 7. The flow diagram is entered on the upper left of the page with the "Take Data" button that has the name `btTakeData`. The shaded boxes represent important classes in the program that are described in **Table 4**:

Table 4 : Classes used during Main Data Acquisition Loop

Class Name	Description
`CBias`	This represents the mixer bias supply, and is responsible for mapping a specific mixer to an analog channel.
`CDataAccess`	This is the main generic database handler. Specific recordsets are retrieved by initializing the class with a set of constant parameters.
`CdbStoreData`	This "child" of `CdataAccss` is a specific database handler that maps field names to generic names for use by the program. The field mapping is a function of the type of measurement, which is passed to the class as a parameter of `bInit`. Note that two instances of this class are instantiated into objects: one is used to save the bias point data, the other to store the swept measurement data.
`CNI16AnOut`	This represents the analog card used for outputting the commanded mixer bias voltage. This class really should be encapsulated into `CBias`.
`CNoiseAnal`	Encapsulates the code required to calculate receiver and mixer noise temperatures.
`CSISDevice`	This class holds the characteristics of the SIS Mixer device, such as the commanded bias voltage, which is a function of the measurement loop index and the number junctions in the device.
`CSpecAnal`	Sets the IF frequency of the measurement by controlling the frequency of the spectrum analyzer with it's IF output connected to either the square law detector or the power meter.
`CTrgSqrLaw`	Responsible for controlling the chopper wheel and returning noise powers (as measured by the square law detector) when the receiver sees the hot and cold loads.

The main measurement loop is highlighted with bold lines. Some important functions are omitted from Figure 7 for clarity, such confirming that measurement records don't already exist for this measurement.

The important module `MeasureResults`, highlighted in bold in Figure 7 near the title block, is responsible for selecting the test equipment required to perform the measurement and is responsible for filling the graph array with data. `MeasureResults` program flow is charted in Figure 8. The class `CTrgSqrLaw` is responsible for controlling the chopper and returning the hot load and cold noise powers (this module currently returns only the Y-factor, so additional coding is required to return the individual average noise powers). `CSpecAnal`, the spcctrum analyzer instrument class, controls the measurement frequency. A new class, `CNoiseAnal`, calculates both receiver and mixer noise temperatures given the measured noise powers.

The mixer temperature calculation requires replacing the mixer's IF output with a hot and cold load and measuring the resulting noise powers. In the case of the integrated mixer/amplifier design, this will measure the noise temperature of the warm IF plate, referred to a reference point inside the Dewar. For mixers without integrated amplifiers, this measures the noise performance of the IF amplifier located in the Dewar.

It is unclear how frequently the IF amplifier noise performance should be measured, but initially it will be measured once at the beginning of a measurement scan, and whenever the IF frequency is changed. Measuring this too

frequently can significantly slow the results, because hundreds of milliseconds are required to reliably switch the Radiall coax switch to each switch position. Since these measurements only determine the optimum operating point and not absolute noise temperatures, this approach seems like a reasonable time *vs.* accuracy tradeoff.

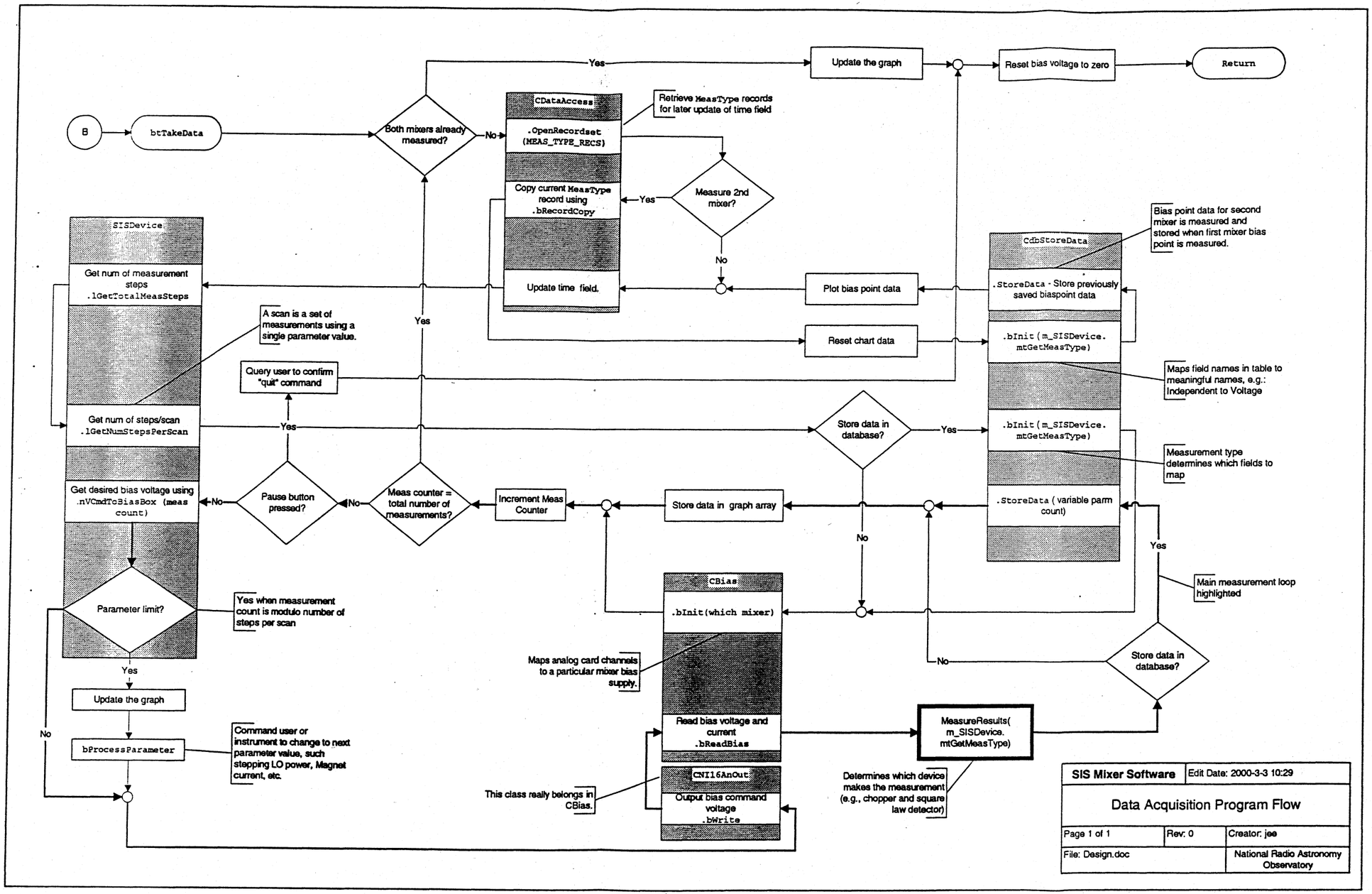

Figure 7: Program Flow for Main Measurement Loop

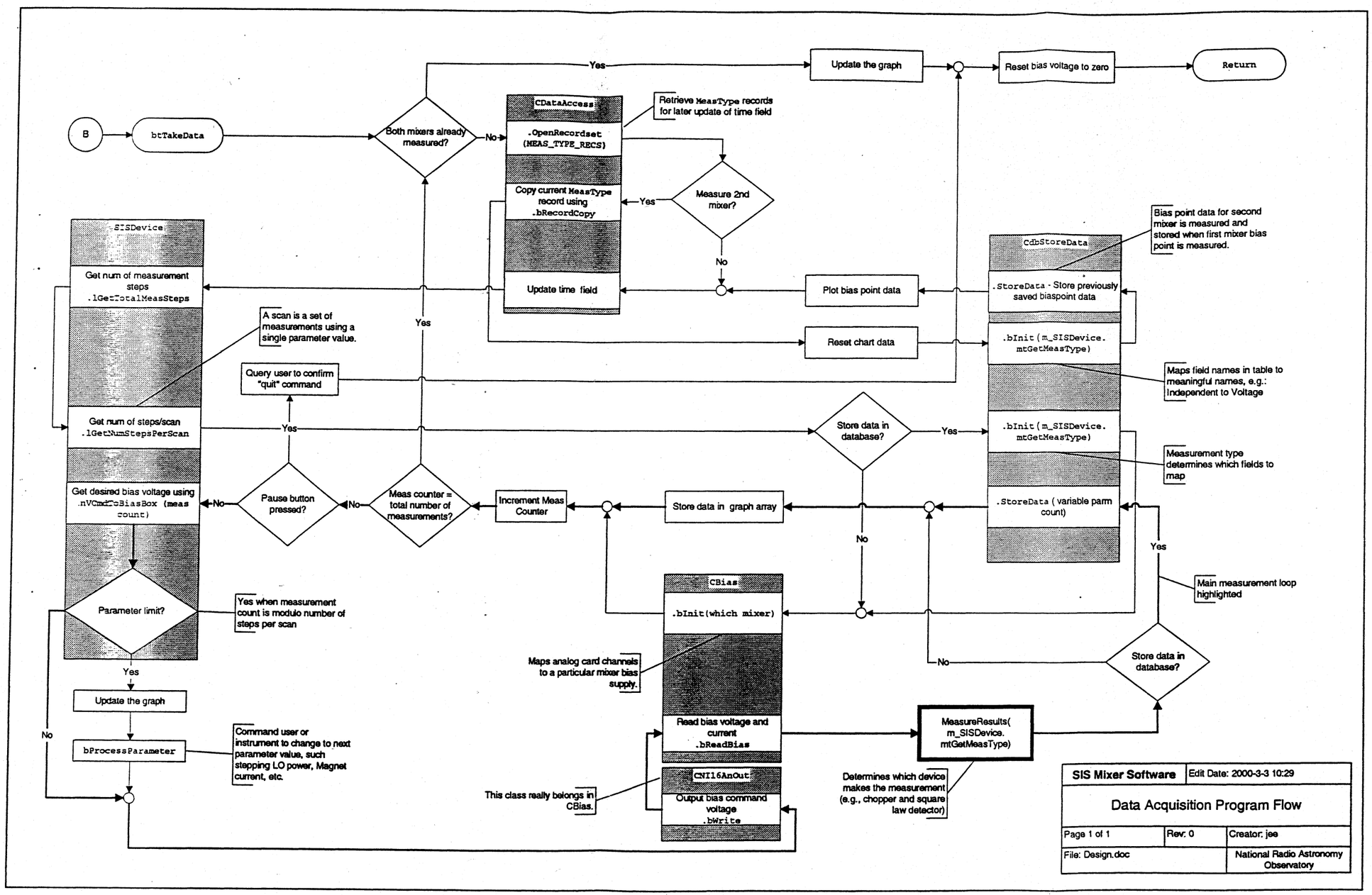

Figure 7: Program Flow for Main Measurement Loop

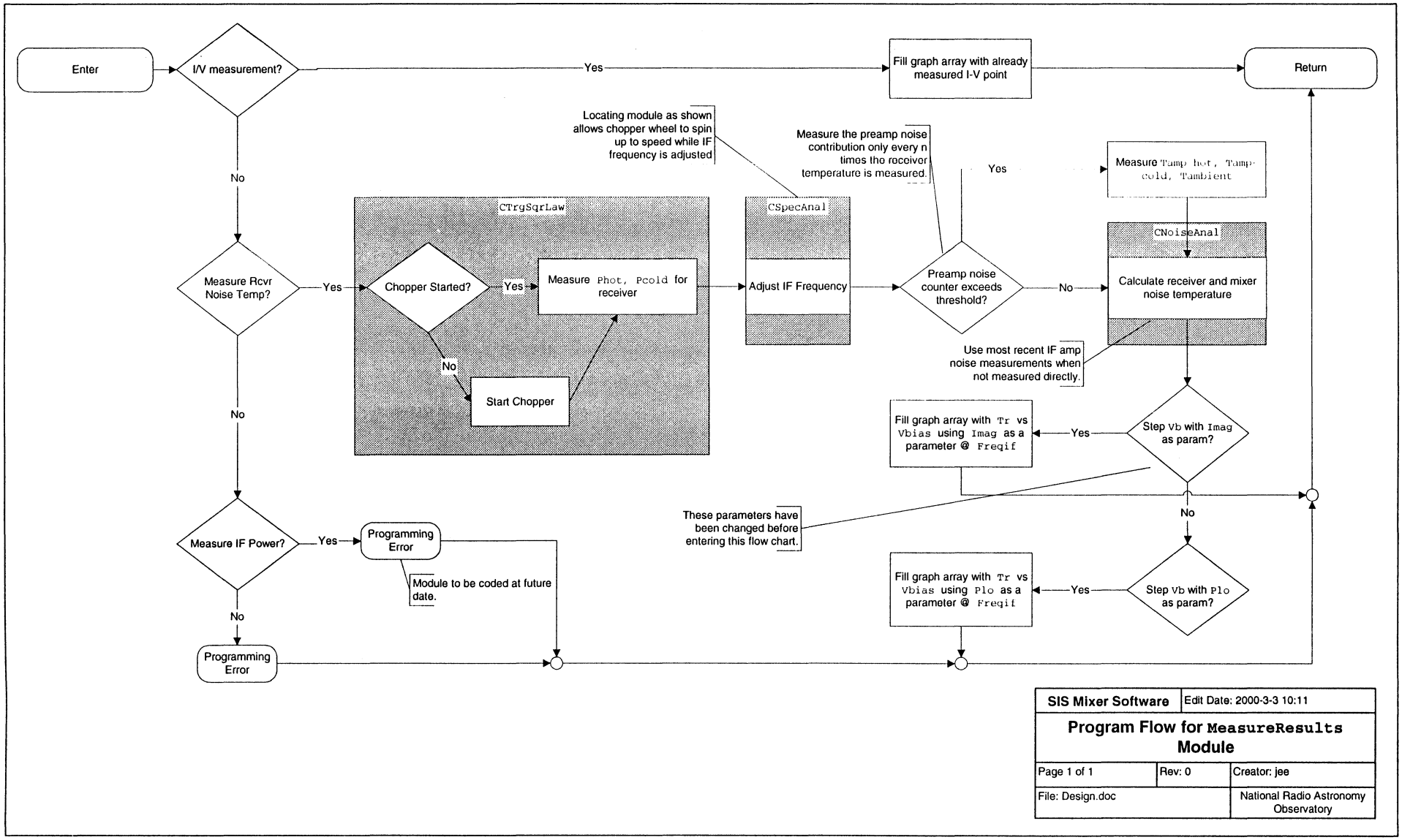

Figure 8: Program Flow for `MeasureResults` Module

Attachment 2: Database Definitions

7. Database

Measurement data is stored in a Microsoft Access database file. The tables are designed to minimize data redundancy by using relational design concepts as defined in Section "8 Table Relationships".

8. Table Relationships

To minimize redundancy, data is spread across several tables as shown in the relationship diagrams in Figure 12 and summarized in Table 5. Details of the relationships follow.

Table 5: Database Table Summary

Table	Description
`SIS Mixer`	Top-level table that contains fields which characterize the mixer undergoing testing
`MeasType`	The type of measurement (such as I-V curve, Noise temperature vs. If frequency, etc) is stored in this table
`Meas`	Holds static data (data that doesn't significantly change during a measurement) recorded during each measurement
`Data`	Contains data that changes for each measurement
`SettingsDefault`	Contains standard setup values for each type of measurement
`SettingsActual`	Stores the settings actually used during a measurement

The top-level table (`SIS Mixer`) contains fields that characterize the mixer undergoing testing, and related tables contain details about each measurement. The *type* of measurement and its associated data is stored in the table `MeasType`, which is linked to `SIS Mixer` though the `Meas ID` key field. The table `Meas` holds static data (data that doesn't significantly change during a measurement) recorded during each measurement. Data that changes for each measurement is stored in another table called `Data`. The table `Data` includes the following fields:

1. A key field (`MeasRecKey`) that links the `Data` records to those in the table `Meas`,
2. a parameter field (`Parameter`), which retains the value being stepped, such as magnet current,
3. a field to record the independent quantity being changed (`Independent`), such as bias voltage,
4. and three generic fields (`Dependent1` to `Dependent4`) to record dependent variable results, such as
    - mixer noise power using a hot load,
    - mixer noise power using a cold load, and
    - bias current.

In the table `Data`, the independent field, parameter field, and dependent fields are generic, single-precision numbers and represent different quantities depending on the measurement. The definition of these generic fields for a particular measurement is located in the general measurement description table `MeasTypeDefs`. The field `Meas Type` in the table `MeasType` contains the measurement type number corresponding to a specific measurement. This structure allows the addition of new measurement types by entering new database records in the `MeasTypeDefs` table, which helps minimize software changes.

Instrument settings for a particular measurement (such as voltage limits and the number of steps) are stored in two tables. The table `SettingsDefault` contains standard setup values for each type of measurement. The information in this table is keyed to the field `TypeNum` in the measurement type definition table `MeasTypeDefs`. The settings actually used during a measurement are stored in the table `SettingsActual`, and that table is keyed to the `Data Key` field in the `MeasType` table, which allows these settings to be recalled for each measurement. The structure of the actual and default setting tables are identical.

9. *Table Relationship Changes for Noise Temperature Acquisition*

The most significant change alters the relationship between the generic results table `Data` and the table `MeasType`, which defines the type of measurement requested. `Data` is presently linked to `MeasType`, but because there can be multiple records in `Data` for a single `Meas` record, a new key field has been entered into `Meas`, called `DataKey`, and all the pointers in `Data` have been reset to link to that key field `DataKey`.

Figure 11 shows how records are created and tables updated during a typical measurement. A new record is written to the table `SIS Mixer` whenever new mixer information is entered. Likewise, a new record is written into the table `MeasType` whenever a new measurement is defined for this mixer. When the measurement is actually started, parameters that change slowly, such as LO frequency and IF frequency, are recorded in the table `Meas` while parameters that change rapidly, such as receiver temperature and bias point, are entered into the generic table `Data`.

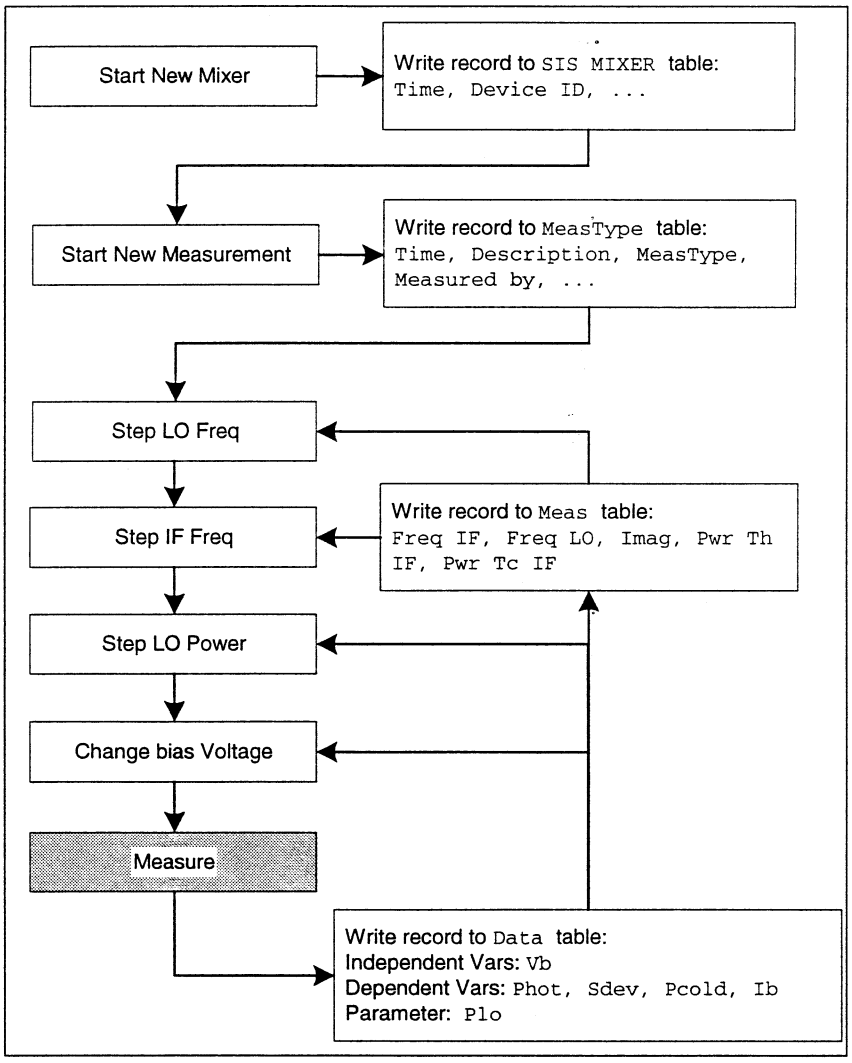

Figure 11: Database Records Created During a Measurement

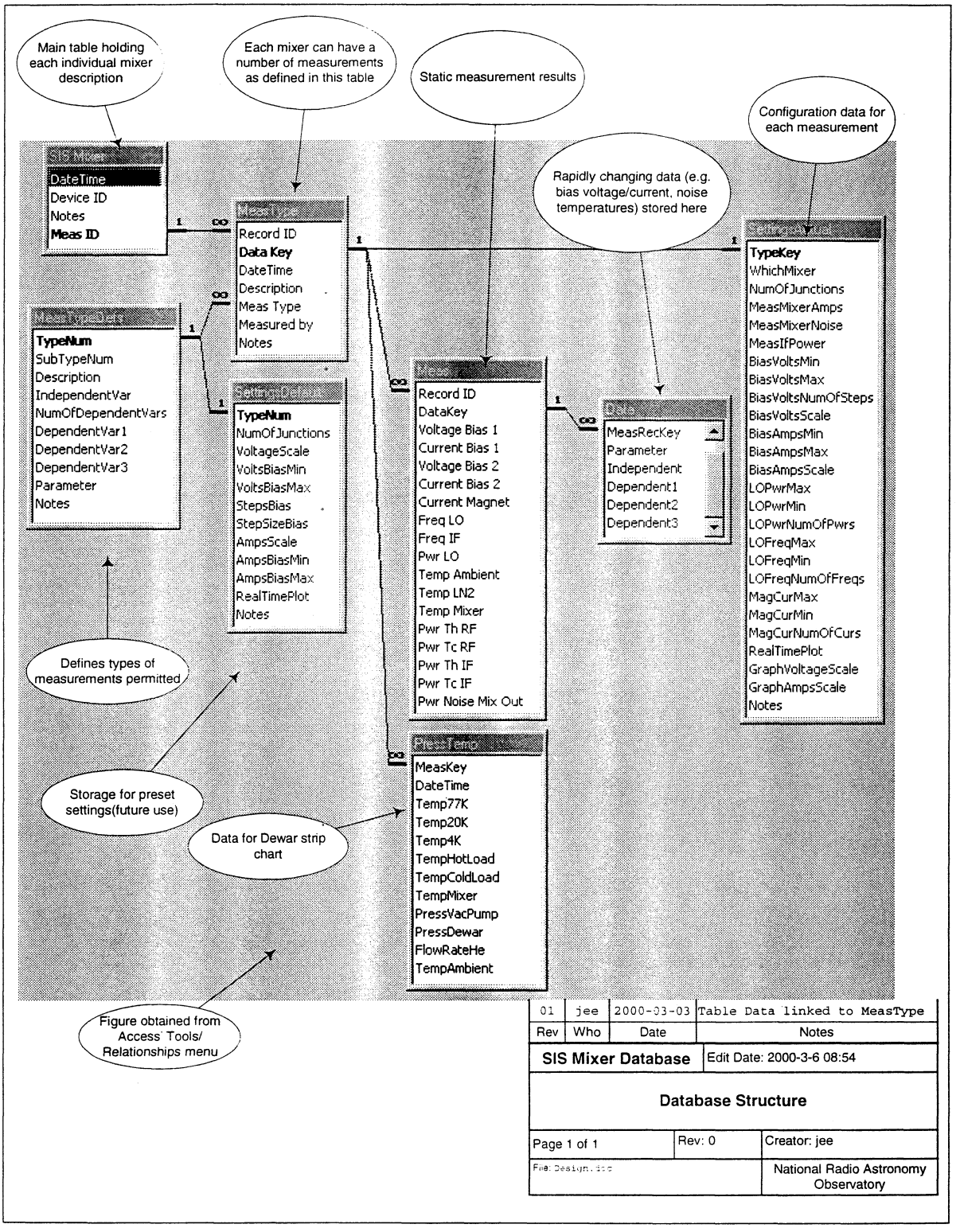

Figure 12: Database Table Relationships