NASA/TM–1999-208755

Microphone Array Phased Processing System (MAPPS) Version 4.0 Manual

Michael E. Watts, Marianne Mosher, Michael Barnes, and Jorge Bardina

March 1999

NASA/TM–1999-208755

Microphone Array Phased Processing System (MAPPS) Version 4.0 Manual

Michael E. Watts and Marianne Mosher
Ames Research Center, Moffett Field, California

Michael Barnes and Jorge Bardina
Caelum Research Corporation, Ames Research Center, Moffett Field, California

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

March 1999

Available from:

NASA Center for AeroSpace Information
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
(703) 487-4650

CONTENTS

MICROPHONE ARRAY PHASED PROCESSING SYSTEM (MAPPS) VERSION 4.0 MANUAL

Michael E. Watts, Marianne Mosher, Michael Barnes,[*] and Jorge Bardina[*]

Ames Research Center

SUMMARY

A processing system has been developed to meet increasing demands for detailed noise measurement of individual model components. The Microphone Array Phased Processing System (MAPPS) uses graphical user interfaces to control all aspects of data processing and visualization. The system uses networked parallel computers to provide noise maps at selected frequencies in a near real-time testing environment. The system has been successfully used in the NASA Ames 7- by 10-Foot Wind Tunnel.

INTRODUCTION

Modern aircraft have improved acoustically to the point that the sound generated by model scale aircraft tested in wind tunnels is at or below the background noise of the wind tunnel for many conditions. Acoustic test techniques are improving to meet the challenge of distinguishing the sound generated by the model from the noise of the test environment. In order to investigate future aircraft noise reduction techniques, the acoustic characteristics of each aircraft model component must be known. No longer is it sufficient to measure noise with a single microphone in a semi-anechoic environment and determine only the overall noise change. Current, and future, noise reduction thrusts are to measure and influence the individually smaller but globally important noise generators such as flap edges and gear struts. We now need to be able to say how the implementation of a noise reduction technique affects the component acoustic characteristics as well as the overall sound level.

This requirement for detailed knowledge of acoustic sources and low signal to noise ratio led to the development and application of microphone array technology to wind tunnel testing. Important factors contributing to the usability and usefulness of arrays in testing environments today include ease of use in processing and visualization interfaces, efficient handling of large quantities of data, speed of obtaining calibrated results, and display of results in an understandable manner. The use of point and click interfaces provide intuitiveness and ease of use in modern systems. This is a desirable goal as it reduces training time and thus increases the number of users who use the system. The combination of large numbers of microphones and the desire to process a large number of frequencies have resulted in

[*] Caelum Research Corporation, Ames Research Center, Moffett Field, CA 94035-1000.

the production of large files for each test point. Thus the network transfer of, and disk access to, data files becomes an issue when considering the bottlenecks in system throughput. Traditional piston phone calibration techniques do not take into account installation or directionality effects. Additionally, the individual calibration of large numbers of microphones is time consuming and tedious. Thus the calibration of array microphones is an important aspect in designing an array system able to produce calibrated results in a timely manner. Array systems not only produce large quantities of raw digital data, but large quantities of processed data. The ability to view and assimilate this data efficiently is even more important than producing the results. After all, if you acquire the data but it is never used, then all the time and money spent in development and testing are wasted.

The requirements mentioned above led to the development of the Phased Microphone Array Technology (PMAT) system. The PMAT comprises two parts: 1) instrumentation and data digitization and 2) data processing and visualization. The Microphone Array Phased Processing System (MAPPS) comprises the second part of PMAT. This paper will discuss the implementation of these requirements into the MAPPS.

The paper is organized to describe first the general and then the specific concepts. An overall system description is presented, followed by general data considerations. Detailed descriptions of the MAPPS functions and interfaces are then presented. Finally, the general utility programs used by MAPPS are described.

SYSTEM DESCRIPTION

MAPPS was developed as part of the PMAT system that encompassed signal measurement, analog to digital conversion, data storage, data processing, and results visualization. The MAPPS begins at the end of the data acquisition and storage and ends with the processed data visualization. This system is designed to be versatile and robust in its treatment of variable numbers of microphones, number and locations of processes, versatile calibrations, and visualization requirements. This versatility is designed into the system to provide for alternative positions if components fail. These component failures will result in degraded results but will still be sufficient to provide researchers with information to meet their needs. Ease of use of the system was also a cornerstone of the design constraints. A point and click graphical interface to the processing and visualization codes was thus developed. This point and click environment will allow a minimally trained researcher to operate the system. The system is designed so that the user may concentrate on research, testing, and data interpretation instead of data and file manipulation.

An operational design goal for the MAPPS was to provide sufficient results in near real time to allow the test director and researcher to make next run content decisions. The first operational test of the MAPPS was in a recent Flap Edge III test using a 100-element microphone array in the NASA Ames 7- by 10-Foot Wind Tunnel. The system had a 9-minute cycle from end of data acquisition to showing results on screen for 166 frequencies with 400 averages and a frequency resolution of 150 Hz. This cycle time was sufficient to obtain results from one point for each run condition and to allow the test director to make model change and run condition decisions for the next run. Another operational design consideration was to have all the data processed and ready for examination by the next day. The ability to batch process multiple data points was also demonstrated at this test.

There are three main elements of MAPPS: 1) processing control interface, 2) processing software, and 3) visualization software. Each element can be treated as a separate entity that is interrelated and interconnected by data files. Figure 1 shows the interrelationship of these elements with process control and data flow. This section describes the computing environment, general data considerations, and the three main elements of MAPPS.

Computing Environment

MAPPS programs currently run on Unix-domain systems that support X Windows. Unlike most operating systems, Unix does not require its users to accept any particular user interface. The X Window System is a network transparent windowing system that runs on a wide range of computing and graphics machines.

The execution of multiple tasks in different machines requires user privilege by the system and user on all machines. In addition, each machine must be a trusted host by the control machine. The ‘xhost’ program can be used to add host names or user names to the list on the control machine to allow connections to the X server. This provides a rudimentary form of privacy control and security. It is only sufficient for a workstation (single user) environment, although it does limit the worst abuses. Environments which require more sophisticated measures should implement other mechanisms.

The X Window System also has a number of severe vulnerabilities. A form of security is provided by using a secure shell (ssh). The program ‘ssh’ is used to securely access another computer over a network. It provides strong authentication and secure communications over unsecured channels. It is intended as a replacement for ‘rlogin’, ‘rsh’, ‘rcp’, and ‘rdist’. It can also replace telnet in many cases. With ‘ssh’, you can create secure remote X sessions which are transparent to the user. If a remote site does not support ‘ssh’, a nonsecure fallback mechanism ‘rsh’ is included.

In summary, it is recommended to access all machines of the system through secure channels and execute the Control program from a trusted host.

Data Considerations

Coordinate Systems

Special consideration was given to the design of the coordinate systems and their implementation to ensure the greatest flexibility for test setups. There are three coordinate systems used in MAPPS: array, model, and wind tunnel. MAPPS converts all locations into the wind tunnel coordinate system internally before processing. Transformation vectors and rotation matrices are included in the raw data file to convert from model and array to the wind tunnel coordinate system. The convention for the definition of the transformation matrix is for conversion from the local to the wind tunnel coordinate system. All coordinate systems are right-handed, Cartesian coordinate systems expressed in inches. The coordinate systems are shown in figure 2.

Wind Tunnel Coordinate System: The Wind Tunnel Coordinate System (x_{wt}, y_{wt}, z_{wt}) must have the x-axis aligned with the flow. The origin can be placed anywhere in the wind tunnel with the x-axis pointing downstream or upstream. The x-axis direction is indicated by the variable mtunxdir contained in the raw data file. Mtunxdir has a value of ‘1’ for the x-axis pointing downstream and ‘-1’ for the x-axis pointing upstream. The wind tunnel coordinate system must be defined before the beginning of a test and stays fixed for the duration of that test since it is the reference for all other coordinate systems used during that test.

Array Coordinate System: The Array Coordinate System (x_a, y_a, z_a) is tied to the array and moves with the array. The simplest way to organize the array coordinate system is to make the center of the array the center of the coordinate system. The z-axis is pointed directly out of the array into the measurement area and perpendicular to a planar array. When the array is viewed from the front, the x-axis points to the right and the y-axis points up. This simplifies the positioning of the microphones and the display of the microphone locations in the processing control interface.

A spherical coordinate system (r_a, θ_a, ϕ_a) is also used for calibration of the array. The transformation to the spherical coordinate system is given by equation (1)

$$\begin{aligned} r_a &= \sqrt{x_a^2 + y_a^2 + z_a^2} \\ \theta_a &= \cos^{-1}\left(\frac{z_a}{r_a}\right) \\ \phi_a &= \operatorname{atan}2(y_a, x_a) \end{aligned} \tag{1}$$

The transformation back to Cartesian coordinates is given by equation (2)

$$\begin{aligned} x_a &= r_a \cdot \sin(\theta_a) \cdot \cos(\phi_a) \\ y_a &= r_a \cdot \sin(\theta_a) \cdot \sin(\phi_a) \\ z_a &= r_a \cdot \cos(\theta_a) \end{aligned} \tag{2}$$

Model Coordinate System: The Model Coordinate System (x_m, y_m, z_m) is tied to the model and moves with the model. If the ‘mview’ program will be used to view the results of array processing, the model coordinate system must be oriented so that the z-axis is perpendicular or nearly perpendicular to the scan surface. Like the array coordinate system, the transformations of the model coordinate system to the wind tunnel coordinate system must be defined before the start of the test. If the model will be rotated, the transformations will be easiest to organize if the center of the model coordinate system lies on the model rotation axis and a coordinate axis is aligned with the model rotation axis.

When a source is used to calibrate the array, the calibration source coordinate system can be considered a model coordinate system. The source calibration coordinate system (x_c, y_c, z_c) is tied to the calibration source. The simplest way to organize the calibration source coordinate system is to make the center of the source the center of the coordinate system and orient the z-axis directly out of the source into the measurement area. When the source is viewed from the front, the x-axis points to the right and the y-axis points up.

A spherical coordinate system (r_c, θ_c, ϕ_c) is also used for calibration of the source. The transformation to the spherical coordinate system is given by equation (3)

$$\begin{aligned} r_c &= \sqrt{x_c^2 + y_c^2 + z_c^2} \\ \theta_c &= \cos^{-1}\left(\frac{z_c}{r_c}\right) \\ \phi_c &= \operatorname{atan} 2(y_c, x_c) \end{aligned} \tag{3}$$

The transformation back to Cartesian coordinates is given by equation (4)

$$\begin{aligned} x_c &= r_c \cdot \sin(\theta_c) \cdot \cos(\phi_c) \\ y_c &= r_c \cdot \sin(\theta_c) \cdot \sin(\phi_c) \\ z_c &= r_c \cdot \cos(\theta_c) \end{aligned} \tag{4}$$

Basic Transformation: In general, a series of several translations and rotations are involved in any specific transformation between coordinate systems. This sequence of translations and rotations can be combined into a standard coordinate transformation. The translation vector and transformation matrix must be determined for each coordinate system to be used in any test.

To transform from a coordinate system other than the wind tunnel coordinate system to the basic wind tunnel coordinate system, first multiply by the transformation matrix and then add the translation vector

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{wt}} = \left\{ \begin{pmatrix} t1 & t2 & t3 \\ t4 & t5 & t6 \\ t7 & t8 & t9 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{o}} \right\} + \begin{pmatrix} xc \\ yc \\ zc \end{pmatrix}_{\mathrm{wt}} \tag{5}$$

The translation vector is the vector from the origin of the basic wind tunnel coordinate system to the origin of the other coordinate system. This vector is measured in the basic wind tunnel coordinate system in inches. The elements of the rotation matrix are the cosines of the angles between the coordinate axis vectors in the two coordinate systems. For simple rotations about one coordinate axis, the elements of the rotation matrix can easily be determined. For more complicated rotations, the matrix can be determined by a sequence of simpler rotations. For all cases the elements are defined as follows:

$$\begin{aligned}
t1 &= \cos\left(\angle\left(\vec{x}_o, \vec{x}_{wt}\right)\right) = \frac{\vec{x}_o \cdot \vec{x}_{wt}}{\left\|\vec{x}_o\right\| \cdot \left\|\vec{x}_{wt}\right\|} \\
t2 &= \cos\left(\angle\left(\vec{y}_o, \vec{x}_{wt}\right)\right) = \frac{\vec{y}_o \cdot \vec{x}_{wt}}{\left\|\vec{y}_o\right\| \cdot \left\|\vec{x}_{wt}\right\|} \\
t3 &= \cos\left(\angle\left(\vec{z}_o, \vec{x}_{wt}\right)\right) = \frac{\vec{z}_o \cdot \vec{x}_{wt}}{\left\|\vec{z}_o\right\| \cdot \left\|\vec{x}_{wt}\right\|} \\
t4 &= \cos\left(\angle\left(\vec{x}_o, \vec{y}_{wt}\right)\right) = \frac{\vec{x}_o \cdot \vec{y}_{wt}}{\left\|\vec{x}_o\right\| \cdot \left\|\vec{y}_{wt}\right\|} \\
t5 &= \cos\left(\angle\left(\vec{y}_o, \vec{y}_{wt}\right)\right) = \frac{\vec{y}_o \cdot \vec{y}_{wt}}{\left\|\vec{y}_o\right\| \cdot \left\|\vec{y}_{wt}\right\|} \\
t6 &= \cos\left(\angle\left(\vec{z}_o, \vec{y}_{wt}\right)\right) = \frac{\vec{z}_o \cdot \vec{y}_{wt}}{\left\|\vec{z}_o\right\| \cdot \left\|\vec{y}_{wt}\right\|} \\
t7 &= \cos\left(\angle\left(\vec{x}_o, \vec{z}_{wt}\right)\right) = \frac{\vec{x}_o \cdot \vec{z}_{wt}}{\left\|\vec{x}_o\right\| \cdot \left\|\vec{z}_{wt}\right\|} \\
t8 &= \cos\left(\angle\left(\vec{y}_o, \vec{z}_{wt}\right)\right) = \frac{\vec{y}_o \cdot \vec{z}_{wt}}{\left\|\vec{y}_o\right\| \cdot \left\|\vec{z}_{wt}\right\|} \\
t9 &= \cos\left(\angle\left(\vec{z}_o, \vec{z}_{wt}\right)\right) = \frac{\vec{z}_o \cdot \vec{z}_{wt}}{\left\|\vec{z}_o\right\| \cdot \left\|\vec{z}_{wt}\right\|}
\end{aligned} \qquad (6)$$

Where the notation $\angle\left(\vec{a}, \vec{b}\right)$ means the angle between the first axis $\vec{a}$ and the second axis $\vec{b}$.

To transform from the basic wind tunnel coordinate system to any other coordinate system, the coordinates will first be translated by a vector and then rotated by the transpose of the rotation matrix,

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{o}} = \begin{pmatrix} t1 & t4 & t7 \\ t2 & t5 & t8 \\ t3 & t6 & t9 \end{pmatrix} \cdot \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{wt}} - \begin{pmatrix} xc \\ yc \\ zc \end{pmatrix}_{\mathrm{wt}} \right\} \qquad (7)$$

Test-specific algorithms must be defined before testing and implemented in the acquisition software such that the correct translations and rotations from other coordinate systems to the wind tunnel coordinate system are saved in the raw data file.

Equations (5) and (7) are implemented in the processing as two subroutines, *o2wt* and *wt2o*. With these two subroutines and the transformation vector and matrix, coordinates can be transformed back and forth between coordinate systems.

Array Coordinate System Transformations: While the wind tunnel coordinate system must remain fixed during a test, the array and its attached coordinate system may be moved during a test. The processing software takes account of translations and/or rotations of the array without any change to the processing software as long as the data file includes the correct transformation information. All transformation distances are stored in inches as the units. If the array moves, there are now three distinct coordinate systems relating to the array, the wind tunnel coordinate system, the reference array coordinate system and the absolute array coordinate system. The data file contains locations for storing information about all three coordinate systems and the transformations between them.

The transformations between the wind tunnel and absolute array coordinate systems are:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{wt}} = \left\{ \begin{pmatrix} ta1 & ta2 & ta3 \\ ta4 & ta5 & ta6 \\ ta7 & ta8 & ta9 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{a}} \right\} + \begin{pmatrix} xac \\ yac \\ zac \end{pmatrix}_{\mathrm{wt}} \tag{8}$$

and

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{a}} = \begin{pmatrix} ta1 & ta4 & ta7 \\ ta2 & ta5 & ta8 \\ ta3 & ta6 & ta9 \end{pmatrix} \cdot \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{wt}} - \begin{pmatrix} xac \\ yac \\ zac \end{pmatrix}_{\mathrm{wt}} \right\} \tag{9}$$

If the array moves between data points, the vector and/or rotation matrix will also change. The vector (center of the array coordinate system measured in the wind tunnel coordinate system) is stored in the data file in the attribute arrayabsxyzwtc. This attribute is stored in inches as:

$$\text{arrayabsxyzwtc} = (xac, yac, zac)_{wt} \tag{10}$$

The rotation matrix is stored in the raw data file in the attribute arrayabsrotwtc as a vector.

$$\text{arrayabsrotwtc} = (ta1, ta2, ta3, ta4, ta5, ta6, ta7, ta8, ta9)_{wt} \tag{11}$$

Storing rotation matrices as linear vectors instead of two-dimensional arrays simplifies tracking the variables and writing programs in multiple computer languages with different array conventions. In order for the processing program to work correctly, these attributes defining the transformations between the wind tunnel and absolute array coordinate systems must be correctly stored in the data file.

For a movable array, one location must be selected as the reference for a given test. The transformations between the wind tunnel coordinate system and the reference array coordinate system are given by:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{wt}} = \left\{ \begin{pmatrix} tar1 & tar2 & tar3 \\ tar4 & tar5 & tar6 \\ tar7 & tar8 & tar9 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{a,r}} \right\} + \begin{pmatrix} xarc \\ yarc \\ zarc \end{pmatrix}_{\mathrm{wt}} \tag{12}$$

and

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{a,r}} = \begin{pmatrix} tar1 & tar4 & tar7 \\ tar2 & tar5 & tar8 \\ tar3 & tar6 & tar9 \end{pmatrix} \cdot \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{wt}} - \begin{pmatrix} xarc \\ yarc \\ zarc \end{pmatrix}_{\mathrm{wt}} \right\} \tag{13}$$

The vector to the center of the reference array coordinate system measured in the wind tunnel coordinate system is stored in the data file in the attribute arrayrefxyzwtc.

$$\text{arrayrefxyzwtc} = (xarc, yarc, zarc)_{wt} \tag{14}$$

Equation (12) is implemented in the subroutine *o2wt*. Equation (13) is implemented in the subroutine *wt2o*. The rotation matrix is stored in the data file in the attribute arrayrefrotwtc as a vector.

$$\text{arrayrefrotwtc} = (tar1, tar2, tar3, tar4, tar5, tar6, tar7, tar8, tar9)_{wt} \tag{15}$$

If the array never moves during a test, this transformation is the same as the one for the absolute array coordinate system.

The following example of an array on a traverse shows how to implement the array coordinate system and how to determine all of the related inputs to the processing program. Let the wind tunnel coordinate system be defined with the x-axis pointing downstream and the origin at the floor level of the wind tunnel in the center. The wind tunnel z-axis points toward the ceiling and the y-axis points to a side wall to form a right handed coordinate system. Suppose the array is mounted on a traverse with the center of the array 6 feet above the wind tunnel floor and 10 feet to the side of the center of the tunnel, the array is aimed 30° up from a vertical setting and the traverse allows the center of the array to range from 5 feet in front of the center to 25 feet behind the center of the wind tunnel. Let the array coordinate system be defined with the center at the center of the array, the z-axis pointing out into the wind tunnel, the x-axis pointing downstream, and the y-axis making a right handed coordinate system. Figure 3 shows this configuration with the array reference location at the farthest forward location of the array and the array positioned at 7 feet behind the center of the wind tunnel. In this example, the array transformations are given by the following values:

$$\begin{aligned} \text{mtunxdir} &= 1 \\ \text{arrayabsxyzwtc} &= (84, 120, 72)_{wt} \\ \text{arrayabsrotwtc} &= (1, 0, 0, 0, 0.5, 0.866, 0, -0.866, 0.5)_{wt} \\ \text{arrayrefxyzwtc} &= (-60, 120, 72)_{wt} \\ \text{arrayrefrotwtc} &= (1, 0, 0, 0, 0.5, 0.866, 0, -0.866, 0.5)_{wt} \end{aligned} \tag{16}$$

Model Coordinate System Transformations: Like the array, it is possible for the model to move during a test. The three coordinate systems relating to the model are the wind tunnel coordinate system, the reference model coordinate system, and the absolute model coordinate system.

To transform from the absolute model coordinate system to the wind tunnel system, use the subroutine *o2wt* with equation (17)

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{wt}} = \left\{ \begin{pmatrix} tm1 & tm2 & tm3 \\ tm4 & tm5 & tm6 \\ tm7 & tm8 & tm9 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{m}} \right\} + \begin{pmatrix} xmc \\ ymc \\ zmc \end{pmatrix}_{\mathrm{wt}} \tag{17}$$

The subroutine *wt2o* is used to move from the wind tunnel coordinate system to the absolute model coordinate system with equation (18)

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{m}} = \begin{pmatrix} tm1 & tm4 & tm7 \\ tm2 & tm5 & tm8 \\ tm3 & tm6 & tm9 \end{pmatrix} \cdot \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{wt}} - \begin{pmatrix} xmc \\ ymc \\ zmc \end{pmatrix}_{\mathrm{wt}} \right\} \tag{18}$$

The vector to the center of the absolute model coordinate system in the wind tunnel coordinate system is stored in the data file in the variable modelabsxyzwtc:

$$\text{modelabsxyzwtc} = (xmc, ymc, zmc)_{wt} \tag{19}$$

The rotation matrix is stored in the variable modelabsrotwtc

$$\text{modelabsrotwtc} = (tm1, tm2, tm3, tm4, tm5, tm6, tm7, tm8, tm9)_{wt} \tag{20}$$

If the model will be moved during the test, one location must be selected as the reference location. The transformations between the wind tunnel coordinate system and the reference model coordinate system are:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{wt}} = \left\{ \begin{pmatrix} tmr1 & tmr2 & tmr3 \\ tmr4 & tmr5 & tmr6 \\ tmr7 & tmr8 & tmr9 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{m,r}} \right\} + \begin{pmatrix} xmrc \\ ymrc \\ zmrc \end{pmatrix}_{wt} \tag{21}$$

and

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{m,r}} = \begin{pmatrix} tmr1 & tmr4 & tmr7 \\ tmr2 & tmr5 & tmr8 \\ tmr3 & tmr6 & tmr9 \end{pmatrix} \cdot \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{\mathrm{wt}} - \begin{pmatrix} xmrc \\ ymrc \\ zmrc \end{pmatrix}_{\mathrm{wt}} \right\} \tag{22}$$

The vector to the center of the reference model coordinate system in the wind tunnel coordinate system is stored in the data file in the variable modelrefxyzwtc:

$$\text{modelrefxyzwtc} = (xmrc, ymrc, zmrc)_{wt} \tag{23}$$

The rotation matrix is stored in the variable modelrefrotwtc

$$\text{modelrefrotwtc} = (tmr1, tmr2, tmr3, tmr4, tmr5, tmr6, tmr7, tmr8, tmr9)_{wt} \qquad (24)$$

The following example shows a model that rotates in a wind tunnel. This example demonstrates how to determine all of the related values stored in the raw data file. Let the wind tunnel coordinate system be defined with the x-axis pointing downstream. The wind tunnel z-axis points toward the ceiling and the y-axis points to a side wall to form a right handed coordinate system. The center of the wind tunnel coordinate system is on the floor at the center of the wind tunnel and the center of the turntable in the wind tunnel. The model, a semi-span wing 3 feet long, is mounted vertically on the turntable so that rotating the turntable imparts pitch to the wing. The simplest transformations will occur when the center of the model coordinate system is on the axis of the turntable so that the center will not translate when the wing is pitched. The center of the model coordinate system is placed at the edge of the wing on the pivot line. The model x-axis points toward the back of the wing aligned with the flow when the wing is at 0° pitch. The model z-axis is pointing toward the top side of the wing and the y-axis is pointing along the axis of the wing to make a right handed coordinate system. Figure 4 shows the coordinate systems. In this example, the model transformations are given by the following values:

$$\begin{aligned} \text{mtunxdir} &= 1 \\ \text{modelabsxyzwtc} &= (0, 0, 36)_{wt} \\ \text{modelabsrotwtc} &= (\cos(\alpha), 0, \sin(\alpha), -\sin(\alpha), 0, \cos(\alpha), 0, -1, 0)_{wt} \\ \text{modelrefxyzwtc} &= (0, 0, 36)_{wt} \\ \text{modelrefrotwtc} &= (1, 0, 0, 0, 0, -1, 0, 1, 0)_{wt} \end{aligned} \qquad (25)$$

Calibration Methods

Obtaining the most useful information from an array measurement requires accurate calibrations of the array. For a single free-field microphone, a high quality calibration includes a frequency response calibration with a reference electrostatic actuator, frequent piston phone calibrations, free-field correction, directivity correction, and microphone forebody corrections for in-flow measurements. A complete calibration of an array involves phase and amplitude calibrations, including the installation mounting effects on the acoustic field. In order to accurately locate noise sources, the *in situ* phase response of all microphones in the array must be known; however, the amplitude can be unknown as long as all microphones have similar sensitivity. Accurate source level measurements require accurate array amplitude as well as phase calibrations. The standard piston phone procedure is time consuming for large numbers of microphones and fails to account for the installation or directivity effects on the acoustic measurements. The mounting of the microphones in an array often changes the frequency and directional response of the microphones. A procedure to calibrate the array with a known source can account for these problems. To produce accurate measurements with the array also requires a directivity calibration to account for transducer directivity and installation effects. These effects become increasingly significant with higher frequencies. These effects can be measured by using a calibrated source with the array in many orientations with respect to the source over the expected range of measurement angles.

The calibration methodology developed at NASA Ames Research Center contains several types of calibrations. This method will be referred to as the Ames method and consists of:

Pre Scan:

- Sensitivity
- Individual instrumentation amplitude and phase
- Installation effects
- Free field

Post Scan:

- Directivity
- Density
- Atmospheric attenuation

Many of the types of calibrations contain multiple options. All of the calibrations used in processing are controlled through one dialog in the control interface. The appropriate files containing calibration information must be present and identified in the process setup. Various testing and calibration processing as described in later sections must be performed prior to array processing to define the calibration files.

Sensitivity: The sensitivity calibration sets the level response of each microphone in the array at one frequency or one narrow frequency range. The sensitivity value is used to convert the data from voltage going into the data acquisition system into Pascals used for processing (Pascals/volt). Either a piston phone calibration or a speaker box calibration may be used to generate these values. To process data without the sensitivity calibration, the user must provide a sensitivity file where the slopes have been set to ‘1.0’ for all the channels.

A method to account for drifts in the microphones and electronics on a frequent basis without having to repeatedly piston phone all channels was developed using a calibrated speaker source in an anechoic box. By using the speaker box option, the microphone sensitivities can be measured closer in time to the actual measurements. This speaker method uses the assumptions that the acoustic field produced by the speaker box at the array can be accurately measured and the acoustic field is repeatable. All of the microphones in the array must first be calibrated with a piston phone. Then position the speaker box on the array and acquire a very long and continuous sample of broad band noise through the data acquisition system. All of the microphones in the array and a reference channel must be recorded simultaneously. This establishes a baseline measurement that will be used to adjust the periodic sensitivities.

To calibrate during testing, place the speaker box against the array, then collect a very long and continuous sample of broad band noise through the data system. The microphones in the array and a reference channel must be recorded simultaneously. A subset of microphones near the center of the array, and an appropriate frequency range, is used to adjust the baseline piston phone calibrations. The frequency range should be wide enough to produce statistically accurate results yet narrow enough so that the response of the instrumentation varies an insignificant amount over the frequency range. The number of microphones chosen near the center of the array must be determined to eliminate microphone variations. The program ‘dailysensitivity’ is used to make the corrected sensitivity file. It should be noted that this method makes assumptions about the speaker box quality and stability of the instrumentation. This method also uses a relatively small number of microphones and frequencies to adjust all array microphones at all frequencies.

Instrumentation: The instrumentation calibration is used to correct the nonuniform amplitude and phase characteristics of the instrumentation. Corrections can be made for the frequency response of the microphone diaphragm at atmospheric pressure, microphone preamplifier, microphone power supply, filter, institutional wiring, and microphone diaphragm response to nonatmospheric pressures. Corrections may be selected for each type of instrument individually and may apply to the amplitude, phase, or both. The correction may come from a calibration of the individual instrument or from a representative curve fit for that model of instrument.

The instrumentation calibration uses the assumption that the shape of the frequency response of each instrument does not vary with time. The sensitivity calibration is meant to take into account the shifts up and down in the amplitude response of the system. Before using any calibration correction, each individual instrument component needs to be calibrated and modeled. The microphone diaphragm is calibrated with an electrostatic actuator; all other instruments are calibrated by measuring a transfer function with noise injection. Calibration curves for the individual instruments and the delta pressure response of the diaphragms are made with the MATLAB program ‘MakeMicModel’; representative calibration curves for the instrumentation model type are made with the program ‘MakeRepMicModel’. The calibration curves are stored in the Instrumentation Calibration Files in a netCDF format. All of the corrections are stored as neural net curve fits. The diaphragm response is modeled as a function of wave number. The delta pressure response of the diaphragm is modeled as a function of wave number and pressure. All other instrumentation components are modeled as a function of frequency. All dependent variables in the neural net curve fits are normalized to a mean of zero and a standard deviation of one. The mean and standard deviation of the data used to generate the curve fits are also stored in the calibration file.

Free Field: The free-field correction accounts for the physical response of a microphone in a free field. Application of the correction is selected in the Calibration dialog from the Control Interface. In general, the correction should be off, unless freestanding microphones are used. The correction is a polynomial curve fit that characterizes the curve supplied by the manufacturer for specific microphone model types. The curve fits for B&K 4135 and GRAS TMS140BF are currently in the processing software. The microphone model information contained in the raw data file is used to select the proper curve fit. The polynomial curve fit for other models of microphones must be added to the processing software in the *freefield* subroutine before this free-field correction can be used with those models.

Installation: The installation calibration accounts for the installation effects of the microphones in the array. The response of a microphone is highly dependent on its immediate surroundings. The installation correction may be either a specified constant decibel level or a position and wave number dependent model based on careful measurement and stored in a netCDF format calibration file. To have no installation correction, select the constant installation correction and enter ‘0’ dB. To correct for a pressure doubling due to microphones flush mounted in a surface, select the constant installation correction and enter ‘6’ dB. More complicated installation corrections may be applied with the position dependent file option.

The position dependent file option assumes that the effect of installing microphones in an array can be measured and modeled as a function of wave number and microphone position independent of time. This option is appropriate for an array with complex construction such as mounting microphones recessed behind a porous screen. The array must be calibrated and modeled before using this option.

First calibrate a noise source in an anechoic chamber with a well calibrated microphone. The locations of the microphone must be known in the calibration noise source coordinate system. The calibration noise source coordinate system, (x_c, y_c, z_c), is another form of a model coordinate system. If the calibration source has an axisymmetric or nearly axisymmetric sound field, the z-axis of the calibration noise source coordinate system must be aligned with this acoustic symmetry axis. For example, if the calibration source is a speaker, place the z-axis of the coordinate system along the speaker axis, pointed away from the speaker. The x-axis and y-axis are placed in the plane of the speaker to form a right handed Cartesian coordinate system. To work with a turntable and "seesaw", place the calibration source on the turn table with the y-axis aligned with the pivot of the turn table. The seesaw should rotate about the x-axis of the calibration source. When the measurements are set up with the turntable and seesaw as described above and shown in figure 5, the angle of the seesaw, α_{sd}, the angle of the turntable, β_{sd}, and the distance from the array center to the calibration noise source center, r_{sd}, are related to the position of the microphone in the calibration noise source coordinate system as:

$$\begin{aligned} x_c &= -r_{sd} \cdot \cos(\alpha_{sd}) \cdot \sin(\beta_{sd}) \\ y_c &= -r_{sd} \cdot \sin(\alpha_{sd}) \\ z_c &= r_{sd} \cdot \cos(\alpha_{sd}) \cdot \cos(\beta_{sd}) \end{aligned} \tag{26}$$

A spherical coordinate system is also used with the speaker. The origin of the spherical coordinate system is also at the center of the speaker. The radius, r_c, is measured from the origin at the speaker center, the polar angle, θ_c, is measured from the positive z-axis, and the azimuth angle, ϕ_c, is measured in the x-y plane from the x-axis. The spherical coordinates are used with the curve fitting routines for the speaker acoustic field. The transforms between the Cartesian and spherical coordinates are:

$$\begin{aligned} r_c &= \sqrt{x_c^2 + y_c^2 + z_c^2} \\ \theta_c &= \cos^{-1}\left(\frac{z_c}{r_c}\right) \\ \phi_c &= \operatorname{atan}2(y_c, x_c) \end{aligned} \tag{27}$$

$$\begin{aligned} x_c &= r_c \cdot \sin(\theta_c) \cdot \cos(\phi_c) \\ y_c &= r_c \cdot \sin(\theta_c) \cdot \sin(\phi_c) \\ z_c &= r_c \cdot \cos(\theta_c) \end{aligned} \tag{28}$$

Long time histories of the reference signal and measurement microphone must be simultaneously acquired with the microphone in many positions. The measurements are first processed with the C program 'spdefxferfun' to make transfer functions of the acoustic field referenced to the reference signal. The resulting file is then processed with the MATLAB script 'MakeSpeakModel' to make a neural net model of the transfer functions in the acoustic field of the noise source. This script generates a netCDF file.

Second, the array is calibrated in the anechoic chamber with the calibration noise source. This measurement should be made as close in time as possible to the pistonphone calibration of the array microphones. The positions of all of the array microphones must be known in the calibration noise source coordinate system. When the calibration source is on the seesaw and the array is on the turntable, the array microphone positions can be easily found. Consider the speaker to be in the model coordinate system and the array in the array coordinate system. Consider a temporary wind tunnel coordinate system aligned with the array coordinate system when the turntable is in the neutral position at $\beta_{sd} = 0$. The microphone locations in array coordinates may be transformed into the calibration noise source coordinate system by first using the *o2wt* program with:

$$\begin{aligned} \text{arrayabsxyzwtc} &= (0,0,0) \\ \text{arratabsrotwtc} &= \left(\cos(\beta_{sd}),0,\sin(\beta_{sd})0,1,0,-\sin(\beta_{sd})0,\cos(\beta_{sd})\right) \end{aligned} \tag{29}$$

then applying the *wt2o* program on the result with:

$$\begin{aligned} \text{modelabsxyzwtc} &= \left(0,-r_{sd}\cdot\sin(\alpha_{sd}),r_{sd}\cdot\cos(\alpha_{sd})\right) \\ \text{modelabsrotwtc} &= \left(-1,0,0,0,\cos(\alpha_{sd}),\sin(\alpha_{sd}),0,\sin(\alpha_{sd}),-\cos(\alpha_{sd})\right) \end{aligned} \tag{30}$$

The array measurements are processed by the C program ‘sparrayxferfun’ to generate transfer functions of the microphones in the array. The resulting file is processed with the MATLAB script ‘MakeInstalModel’ to generate a a neural net curve fit for each microphone in the array. The curve fits are stored in a netCDF file.

Directivity: The directivity calibration accounts for the directional response of the microphones as installed in the array. This directivity is different from the directionality of the beam in the phased array processing. The directivity correction is a function of wave number and scan point location. If the directivity correction is activated in the Calibration dialog in the Control Interface, then a netCDF format file containing the directivity curve fit must also be selected.

The directivity correction assumes the directivity response of the array depends on the wave number and direction (θ_a and ϕ_a) and does not depend on distance between the array and the sound source. Before using the directivity correction, the array must be calibrated and a model made of the array directivity. The directivity model is made by first recording measurements with the array of the calibration source at a reference location in the anechoic chamber. The calibration source signal must be recorded simultaneous to the array signals. Next, make measurements with the source in many directions. Process all these directivity points with coordinate transformations that put the array in the array coordinate system and the calibration source in the model coordinate system. Figure 6 shows these coordinate systems for placing the array on the turntable and the calibration noise source on the seesaw. With this configuration, processing may proceed by using the coordinate transformation parameter listed in equations (29) and (30). Run the program ‘dirprennet’ on all the processed directivity measurements to produce files of directivity information. Run these files through the MATLAB program ‘MakeDirModel’ to generate the netCDF file that contains the neural net curve fit for the directivity.

Density: The density correction is a scaling of aeroacoustic sources to the reference conditions. It is assumed that the acoustic pressure radiated by a source is proportional to the ambient static pressure and that the ideal gas law holds.

$$p_{acoustic} \propto \rho_s \tag{31}$$

$$p_s = \rho_s \cdot RT_s \tag{32}$$

The acoustic pressure at the reference and tunnel conditions can be related as:

$$p_{acoustic,ref} = p_{acoustic,tunnel} \cdot \frac{p_{s,ref} \cdot T_{s,tunnel}}{p_{s,tunnel} \cdot T_{s,ref}} \tag{33}$$

$$dB_{ref} = dB_{tunnel} + 20 \cdot \log_{10}\left(\frac{p_{s,ref} \cdot T_{s,tunnel}}{p_{s,tunnel} \cdot T_{s,ref}} \right) \tag{34}$$

The reference conditions are $p_{s,ref}$ =2116.8 pounds per square foot and $T_{s,ref}$ =519° Rankine.

Atmospheric Attenuation: Atmospheric attenuation accounts for excess attenuation of sound waves as they propagate through the atmosphere. This correction has not yet been implemented.

Scanning Surface Definition

MAPPS can scan any surface in space which has a single intersection with a ray projected from the array center. In other words, the scan surface cannot fold back on itself as seen from the array. This limitation is due to the fact that the visualization interface program maps the surface onto a flat plane. The scan surface is defined in model coordinates and inches. This allows the user to define a surface and have that surface move with the model by only specifying the coordinate system transformation from the model to the wind tunnel coordinates. Multilevel grids can be processed but must be displayed with programs other than the current configuration of the MAPPS visualization software.

A predefined scan surface can be read from a Plot3d single zone binary formatted file (ref. 1). This allows for the definition of a complex shaped scan surface. For example, one test had the scan surface follow the dihedral of the model to give a relatively constant distance of the scan surface from the model surface. The user can also define a planar scan surface from the Process Control Interface by defining the upper left, lower left, and upper right corner points and specifying the grid density.

Plane wave processing can also be performed which only scans along directional lines without regard to distance. When plane wave processing is selected, the region to scan is defined by angles instead of a surface.

Data Acquisition Blocks

The amount of raw data required for processing depends on the sample rate, fast Fourier transform (FFT) block size (or required frequency resolution), and number of FFT blocks to average. Normal data processing uses between 100 and 400 FFT averages depending on the relative levels of the background and signal noise levels. This number of averages results in 1 to 5 seconds of data being required for high sample rates with a moderate frequency resolution. Some tunnels have a low frequency oscillation in the flow which requires the tunnel condition measurement systems to average over as long as 20 seconds. Data acquisition for array measurement should span the same period of time as the tunnel condition measurements to obtain good correlation between the two. Taking array data, especially at the higher sample rates, for this 20-second period is prohibitive due to the available memory on the acquisition cards and the disk storage required. To compensate for this time difference the raw data is block decimated in time to store only the amount necessary for array processing yet still be congruent with the tunnel condition measurement. The user specifies the number of samples per data block, the number of seconds to skip between data blocks, and the number of data blocks to acquire to yield an effective time span for data acquisition. The limitation for this method is that it creates "data walls" between acquisition data blocks. For processing, the user specifies the FFT block size (or frequency resolution), the number of FFT blocks per group, and the number of groups. Thus the user can process a subset of the stored data as long as the FFT blocks do not cross the data walls created when the data was digitized. The interface alerts users if they enter combinations which cross these data walls. This decimation method is illustrated in figure 7.

Microphone Health Checking

Array measurements generate a large amount of data from many microphones. This large volume of data makes it inefficient to check each block or channel of microphone data by hand for data quality. Three automated data quality checks were implemented to ensure that only good data were used in the processing. The first is a simple band edge check to indicate clipping in the analog to digital (A/D) process. The second is a check for flat spots in the signal, which indicate clipping in the analog hardware such as amplifiers. The third check is to determine acceptability of the overall levels of the microphones. Given that the primary source of the measured noise is generally near the normal to the array, it can be assumed that all the microphones should see a similar amplitude signal. Using this assumption, a scheme to check for amplitude deviation is used. The block averaged individual microphone spectra are averaged over all the microphones. This signal is then power summed over a specified frequency range to yield a benchmark level. Each individual microphone averaged spectrum is similarly summed. A microphone is determined to be bad if its individual level differs by more than a specified decibel value from the benchmark level. This level change is either an indication of a bad gain setting or that the calibration level in the electronics has slipped or instrumentation has gone bad. The delta decibel check can effectively be removed by setting the decibel level to a high number. This might be necessary if the primary noise source is at an oblique angle to a large array, thus causing a significant variation in sound pressure across the array. Experience has shown that three decibels is a good level for sources normal to the array and six decibels for moderate angle or close sources. Additionally the frequencies for the summations should be within the desired measurement range. The current acceptable limit for processing is that 80% of the FFT data blocks must be considered good. Processing will not complete and the user will be notified if more than 20% of the FFT data blocks are flagged as bad.

Added Gains

Many times during system checkout there is no calibration information available for processing, yet the processing needs to be run. The added gains feature allows the user to generate images that are accurate in location but ambiguous in amplitude by generating gains for each channel which will bring them to a common level. The algorithm for the added gains calculation is the same as that used in the delta decibel health checking. The delta decibel level generated by subtracting the individual microphone frequency sum from the summed microphones frequency sum is considered a channel gain. This has the effect of normalizing the amplitudes to the overall summation level. This feature does not correct for phase differences between channels and ignores all other gains.

Noise Reduction Methods

When the array is used to measure noise in a wind tunnel, the signals the microphones measure consist of sound from any aircraft model plus background noise. Noise contamination due to wind noise over the microphones can be reduced by modifying the diagonal elements of the cross-spectral matrices before scanning. Four methods for modifying the cross-spectral matrices are available. The method labeled “None” will leave the cross-spectral matrix as it is. The “Zero Sub” method sets all the diagonal elements to zero and adjusts the amplitude of the scans to compensate for the zeros in the diagonal. The “Avg Sub” and “Avg Sub 2” methods substitute estimates of the autospectra for each microphone without the uncorrelated background noise. The Avg Sub method replaces the diagonal of the cross-spectral matrix with the average amplitude of the cross spectra (excluding the diagonal value) along a microphone row. The Avg Sub 2 method uses the Avg Sub results and refines the estimate by normalizing with the relative amplitude of the cross-correlation values. If there is uncorrelated background noise, Avg Sub 2 is usually the best method for reducing the noise contamination.

Side Lobe Reduction

Another enhancement method is “Side Lobe Reduction”. Side Lobe Reduction removes any side lobes that show a negative amplitude. This option removes the highest side lobe from a one-dimensional array scan of a point source. This option has not been effective with two-dimensional arrays measuring real sources.

Processing Control Interface

The parallel phased-array processing part of MAPPS can be executed in either batch or interactive mode. The interactive mode is controlled through an X-Motif graphical interface that allows the user to easily set all options used in the processing. The various windows take into consideration the information contained in the raw data files and allow the user to set the proper process control parameters. This prevents processing with inappropriate settings. These option settings are written to an ASCII file that is then read by the Control Process.

Main Window

The Processing Control main window is used to set the processing parameters and access other dialogs via pull down menus. The main window with the menu options is shown in figure 8. Data entry fields are indicated with white boxes and static text fields are shown with gray boxes. Pop up menus are indicated by shaded rectangles. Editable fields will turn red when an unacceptable number is entered. Each field is described below.

Pull Down Menus:

File: This pull down menu allows the user to save and load previous setups of the control interface as well as load the parameters from a specific raw data file.

Display: Pull down menu that allows the user to display information about a specific data file by showing the actual netCDF header information (fig. 9) or a table of information about the instrumentation contained in the specified raw data file (fig. 10).

Parameters: This menu activates the Custom Array, Processing Geometry, Processor Setup, Calibration, Flow Conditions, and Preferences windows described in later sections.

Upper Left Area:

Test, Run, Point, Array Point Number: The raw data file designators set when Load Raw Data File command was executed.

Version Number: The processed data version number used to track processing. This field is used to distinguish between different process settings applied to the same raw data.

Mid Left Area:

FFT Block Size: The number of samples for each FFT to be performed. An approximate number can be entered and it will be rounded to the closest number of $2^l * 3^m * 5^n$

FFT Freq Resolution: Hertz per bin from the FFT determined by (Sample Rate) / (FFT Block Size). An approximate number can be entered and it will be changed to the closest number that meets the FFT Block Size limitations.

FFT Blocks Per Group: The number of averages for each data block available in the raw data based on the FFT Block Size is shown in the Available column. The user enters the desired number in the Requested column. If the Requested column is not an acceptable number then the field will turn red.

Number of FFT Groups: The number of groups to be processed. The maximum number of averages to perform = (FFT Blocks Per Group) * (Number of FFT Groups). The number of data groups contained in the raw data is shown in the Available column and the user enters the desired number in the Requested column. If the Requested column is not an acceptable number then the field will turn red.

Time Span (Sec.): The time spanned by the combination of the above parameters using the skip block methodology defined in a previous section.

Sample Rate: The samples per second with which the data was acquired.

Upper Frequency: The least of the internal and external upper limit filter frequencies, and the analog to digital frequency span.

Lower Frequency: The greatest of the internal and external lower filter frequencies.

Upper Right Area:

Processing Type: Pop up menu that contains the processing type. The only type available in this release is Regular.

Noise Reduction Method: Pop up menu for choosing the noise reduction method as described above.

Side Lobe Reduction: Option to activate the Side Lobe Reduction described above.

Convection Correction: Option to activate convection correction application.

Added Gains: Option to turn on/off added gains.

Time Domain Windowing: Choice of the time domain window to be applied to the time data before the FFT is performed.

Mid Right Area:

Antenna Gains (Use Gains From File or Use Value): Use gains from file will use a sum of preamplifier, power supply, and external filter gains applied to each channel individually. Use Value will ignore the internal gains and apply the specified gain value to all channels.

Bad Microphone List: Any channels listed will be ignored in the data processing. Note that the first channel is channel 1.

Array Chosen: This field indicates if a saved array subpattern has been chosen in the Custom Array window. If Full Array appears then no preset array subpattern has been selected.

Processing Method: Information box indicating which processing method will be used.

Lower Area:

Input Frequencies to be Processed: The desired frequencies to be processed are entered in this field. Single frequencies can be entered separated by a comma. Frequency ranges can be entered by specifying the start frequency and the end frequency with the skip frequency value entered between them in parenthesis. Note that a (1) is every frequency, a (2) is every other frequency, etc. Combinations of single frequency values and frequency ranges can be made in this field. For example "1000(2)2000,5000" will process every other frequency from 1000 to 2000 Hz and 5000 Hz.

Frequencies Processed: This field gives the actual frequencies that will be processed given the limitations of the frequency resolution on the requested frequencies entered.

Number of Frequencies: The total number of frequencies that will be processed.

Bottom Area:

Execute: Button to start interactive execution of the processing software.

Customize Array Window

The Custom Array Window is activated by selecting Array under the Parameters pull down menu. This window (fig. 11) is used to indicate the status of the microphones as indicated by the acquisition system and to select which microphones will be used for processing. The purpose of selecting subsets of microphones (subarrays) is that processing different geometric configurations of microphones yields different frequency and resolution characteristics. For example, if the noise source is a distributed source such as jet noise, selecting a small array of closely spaced microphones will produce better estimates of the source strength than using a large array. The more microphones in an array, the better the array will be at separating sources from background noise; thus if background noise is high, selecting a subarray may not yield beneficial results.

The x and y coordinates in the array coordinate system from the Raw Data File are used to display the microphone array pattern. The array pattern ID, run, and point number are displayed in the title area. Left mouse button clicking on a “microphone” will toggle the microphone status between active and inactive for processing. Right clicking on a microphone will display the channel number in the lower right corner of the window. Changes to microphone status are not registered until “OK”, “Apply”, or “Save...” are clicked. Changes applied to this window may be saved to a file and later reloaded using the “Save...” and “Load...” buttons. Clicking “Cancel” will close the window without applying any changes.

Active microphones used in processing are drawn as solid. User declared inactive microphones selected in this window are drawn as black dashed circles. Variations of the symbols indicate why a microphone will not be used for processing. User declared bad microphones entered from the main window are drawn with a red X through the center of a black dashed circle. A bad channel declared from the data acquisition system that is declared bad in the raw data file is drawn as a red dashed circle. Microphones declared bad in the raw data file and in the user-defined area of the main window are indicated by a red X through the center of a red dashed circle.

Processing Geometry Window

The Processing Geometry Window is activated by selecting Processing Geometry from the Parameters pull down menu. The processing type and scan surface to be used are set in this window. The two choices for processing type are spherical and plane wave and are selected with the processing pop up menu located in the upper half of the window. Figure 12a shows the plane wave processing chosen and the configuration for entering the scan geometry for plane wave processing. The phi and psi angles are all that are required to be entered for plane wave processing since the scanning is performed along radial lines specified by angles from the normal to the array. Phi is the latitudinal angle with $\varphi_a = 90^\circ$ in the direction of the positive y_a-axis. Psi is the longitudinal angle with $\psi_a = 90^\circ$ in the direction of the positive x_a-axis. Phi and psi are shown in relation to the array Cartesian coordinates in figure 12b.

Spherical processing requires the definition of a scan region in space as described in the Scan Surface Definition section. Figure 13 shows the Processing Geometry window in the configuration for entering a scan region defined in a file. Figure 14 shows the window in the configuration for defining a scanning plane in this window.

Processor Setup Window

The Processor Setup Window is activated by selecting Processor Setup from the Parameters pull down menu. The processing distribution configuration is controlled through this window. Details of the data processing are presented in the Parallel Processing Program section. There are four main processes within the data processing software: control, input, parallel processing, and output.

The Processor Setup Window has five sections as shown in figure 15. The first three sections are used to enter the processor name and the executable path to the processing code on that processor for the Control, Input, and Output processes. The Computation Processor Section allows the primary processing to be performed on multiple CPUs across multiple platforms. The user enters the name of the platform, executable path on that platform, and the number of CPUs to use on that platform. One of the platforms that are entered must be designated the lead process. Also specified in this window is the output path for the processed data. The Raw Data and the Scan Geometry are indicated in this window for information purposes.

Calibration Window

The Calibration Window is activated by selecting Calibration from the Parameters pull down menu. This window is used to control all calibration functions. This window has two main areas, one for Pre Scan and one for Post Scan options. The two methods available for Pre Scan calibration are Ames and Boeing 95 and they are selected through the pop up menu at the top of the Pre Scan area. The Boeing 95 method results in a blank Pre Scan area as these calibrations are performed in an external program prior to data processing. The Ames method results in the dialog as shown in figure 16. The three areas of calibration contained in the Pre Scan area are for control of sensitivity, individual instrumentation, and installation calibration. The sensitivity calibration values are input into the program from the ASCII file in the Sensitivity Source File field. The Calibrator or Speaker Box switches indicate the method of generating the values in the Sensitivity Source File.

The calibration “cascade” for the individual instrumentation is controlled through the Individual Instrumentation Calibration area. The cascade is read from left to right using the pop up menus for each instrumentation type. The first column of pop up menus refers to the individual instrumentation and the second column refers to the model type calibration. These columns take precedence in left to right order. For example setting both the Serial # and the Rep. columns to “on” means that the program will first look for a calibration curve for that specific serial number and if it is not found will look for a calibration curve for that model type. If the Serial # is “on” and Rep. is “off”, then the code only looks for Serial # and returns ‘0’ if it is not found and does not progress to the representative fit. Likewise, if the Serial # is “off” and the Rep. is “on”, then the program will only look for the model type representative curve and return ‘0’ if it is not found. The third column of pop up menus contains the choice to apply the amplitude, the phase, or both amplitude and phase calibrations for that instrumentation. The neural net curves are contained in the netCDF format file indicated in the Instrumentation Calibration File field.

There are two options for the installation gains. A constant value to all channels can be applied by selecting the Constant button and entering the value in the text field. A frequency dependent correction can be applied by selecting the From File button and entering the netCDF file name in the text field. If no installation gain is desired then the user should select constant value and enter a ‘0’.

The three Post Scan calibrations are controlled in the lower region of the Calibration Dialog. Each of the Atmospheric Attenuation, Directivity, and Density options has an On/Off pop up menu. The Directivity must have a netCDF file containing the neural net curve fit if it is activated.

Flow Conditions Window

The Flow Conditions Window is activated by selecting Flow Conditions from the Parameters pull down menu. In this window (fig. 17) the flow conditions may be changed to values different from those saved in the raw data file. The raw data file contains the flag wt_data_quality that is set to either good, approximate or absent. The Control Interface checks the wt_data_quality flag when a raw data file is loaded and activates this window if the flow condition data is flagged as approximate or absent. The user can also activate the window from the Parameters menu and override the tunnel condition data that are labeled as good in the raw data file. The values contained in the raw data file are shown in the noneditable fields along the right side of the window with corresponding editable fields next to them where the user can enter the desired values. When the user selects “OK” or “Apply”, a confirmation dialog box appears to ensure that the proper action will be taken. Data entered and passed to the processing code does not change the raw data file values. If it is determined that the values in the raw file need to be changed, then a separate code should be run that changes these variables and updates the data quality flag. The inclusion of this option allows continued data acquisition and processing if the tunnel conditions are not immediately available.

Preferences Window

The Preferences Window is activated by selecting Preferences from the Parameters pull down menu. This window is where general preferences for processing are set and is shown in figure 18. Currently, only the microphone testing parameters are set here. The lower and upper frequency summation limits are the frequency range to be used in the microphone health delta decibel test and the added gains calculation. The delta dB level used to determine if a microphone is deemed good is also entered here.

Processing Software

The parallel processing portion of the MAPPS software has the capability to run in a single workstation or a heterogeneous network of computers. The system processes the data on parallel processors using the Parallel Virtual Machine (PVM) library. The main languages used are Fortran and C. The following sections give a general description of basic PVM information and MAPPS processing software.

Parallel Virtual Machine (PVM)

The Parallel Virtual Machine (PVM, ref. 2) library provides the capability for the program to run in a network of processors. Another newer and improved library, MPI, was considered in the design process, but was not implemented because it did not provide the capability to run heterogeneous processors. PVM allows a heterogeneous network of parallel and serial processors to appear as a single concurrent computational resource, the virtual machine. General information about PVM is found in several sources. A complete overview of PVM is found at http://www.epm.ornl.gov/pvm/pvm_home.html and a complete index for the PVM library is shown at http://www.netlib.org/pvm3. All the information needed to use PVM is also described in the PVM Reference Manual.

The PVM subroutines are intrinsically coupled and used inside the MAPPS processing software. These subroutines add processors to the virtual machine, spawn tasks in each processor, transfer and receive data between processors, and control all the information flow of the virtual machine. PVM has to be previously installed in each processor to be used in the virtual machine. If PVM is not already installed in a given processor, the PVM Reference Manual provides a complete description of how to obtain and install the software.

The environmental variables and aliases PVM_ROOT, PVM_ARCH, pvm and pvmd should be defined in each processor under the account that will execute the MAPPS program. For example:

```
setenv  PVM_ROOT    /usr/local/pvm3
setenv  PVM_ARCH    SGI64
alias   pvm        '$PVM_ROOT/lib/pvm'
alias   pvmd       '$PVM_ROOT/lib/pvmd'
```

It is very convenient to use the same path definition in each processor. PVM_ROOT gives the location of the PVM directory, and PVM_ARCH defines the particular architecture of the processor. PVM_ARCH is defined as SGI64 in the newer Silicon Graphics workstations or servers. Names corresponding to other architectures are listed in the PVM Reference Manual. All executables of the MAPPS processing software should be located or linked to the default location once the PVM software is installed. This must be done in each processor to be used. A convenient way to define these default locations is $PVM_ROOT/pvm3/bin/$PVM_ARCH, or $HOME/pvm3/bin/$PVM_ARCH, if the user does not have the required privilege to copy into the system directories. It is recommended to define the aliases and environmental variables in a shell file executed at login (for example .cshrc) of each processor of the virtual machine. PVM must be installed even if all processes are to be executed on the same CPU.

PVM has to be activated before running the MAPPS software. PVM is started on any of the hosts in the virtual machine by executing the pvm command which is located in the subdirectory $PVM_ROOT/lib. Once the PVM console is started, the MAPPS processing program may be executed. It is also convenient to know that the PVM console may be started and stopped multiple times on any of the hosts without affecting PVM or any other application. A list of PVM commands is obtained by typing "help" on the PVM console. The PVM console stops with the command "quit", and terminates all PVM processes with the command "halt". PVM must be started manually before batch execution but is automatically started when the Process Control Interface is launched.

MAPPS Processing Software

MAPPS processing software is composed of four programs: the Control, the Input, the Parallel Processing, and the Output programs. This division provides the capability to run different aspects of the processing software on separate computers. All the software is controlled with the Control program from a workstation where the user has the required network privileges to access the other computers. The acoustic data is read with the Input program which is executed on a processor that has access to the input and calibration data files. The parallel processing of the acoustic data is run with the Processing program on a set of parallel processors. The results are stored to disk with the Output program. The MAPPS processing software components may be run in a single workstation or spread over several workstations. The general flow of processing information is shown in the MAPPS flow map diagrams in Appendices 1 through 4. Each subroutine includes a detailed description of its function and variables. This information includes a description of each modification, the definition of each input, local, and output variables, and a comment of each processing step inside the program. A general description of the MAPPS programs is given below.

Control Program

The Control program is the main program of the MAPPS processing software. This program is called to initiate processing by either the Control Interface or an executable script. Together with this program, there are two makefiles to compile the subroutines and create an executable code, called Makefile.SGI64 and Makefile.SGI64_debug. The first makefile is for normal execution of the program and the second is for execution in debug mode. Their executable files are ‘mapps_con’ and ‘mapps_con_debug’, respectively.

The main actions in the mapps_con program are:

1. Enroll itself with the PVM daemon (*pvmfmytid*).
2. Read the inputs for processing (*nmlinput*).
3. Configure the virtual machine to run all the programs, start the input program, start the parallel processing programs (*hosts*).
4. Send the required input parameters to the Input program (*pvm*).
5. Modify the units of input parameters to metric units (*rosetta*).
6. Generate the coordinates of the scan plane (*gensp* or *genpl*).
7. Send the required parameters to lead Parallel Processing program (*xferout*).
8. Send all other parameters to lead Parallel Processing program (*nmlsend*).
9. Receive and print end-of-run flags from Input program (*pvm*).
10. Receive and print end-of-run flags from lead Parallel Processing program (*pvm*).
11. Receive and print end-of-run flags from Output program (*pvm*).
12. Exit the PVM daemon (*pvmfexit*).

A typical interactive run in the ‘pmat1’ processor, executing eight parallel tasks in the ‘leonardo’ processor, will produce print statements on the screen similar to the ones shown below.

```
comp_proc(i) = leonardo             524288       1
raw_exec =  /usr/local/pvm3/bin/SGI64/mapps_input
raw_proc =  pmat1
input_tid =     262148        1
comp_exec(i) =  /usr/local/pvm3/bin/SGI64/mapps_proc
comp_proc(i) =  leonardo
```

```
num_cpu(i) =  8
cpu_tids(i) =524289 524290 524291 524292 524293 524294 524295 524296

pvmfconfig:
i info nhost narch dtid speed name
17 1 2 1 262144 1000 pmat1
2 1 2 1 524288 1000 leonardo
End of hosts.F

0 MAPPS program version 4.0 number 25
0 run version number 10 in process.
      Successful completion of control program,
      Parallel processes running on background.
        Successful completion of parallel processes.
      Output process running on background.
        Successful completion of output process.
```

The first group of statements show the executable programs, the processors, and the task ID numbers (tid) generated in the *hosts* subroutine of the control program. The next group of statements shows the current version number that is being processed. The last group of statements is printed to indicate when the control parallel processes and output programs are started and finished.

If an error is detected during execution, an error message is printed showing the name of the subroutine where the error occurred. In particular, a message is printed if too many of the blocks or channels are flagged as bad according to the criteria imposed on the acoustic data in the health matrix calculation.

Once the program is finished, the output file is stored in netCDF format in the location specified in the input control with the variable out_path.

Input Program

The Input program performs the data gathering and memory allocation functions for MAPPS. A temporary netCDF file is created that contains the header information from the input raw time history netCDF file. The new netCDF file will be read later by the Output program to create the output netCDF file.

The main actions in the ‘mapps_input’ program are:

1. Enroll itself with the PVM daemon (*pvm*).
2. Get the task ID of the parent process (*pvm*).
3. Receive the inputs from the control program (*pvm*).
4. Eliminate blank spaces of strings.

5. For processing boeing data, if data_source is ‘boeing’:
   Allocate memory for cross-correlation matrix (*getmem32*)
   Open and read netCDF file (*get_nc_boeing*).
   Read SWTS wind tunnel parameters (*read_nc_swts*).
   Read model and array coordinates (*read_nc_xyz*).
   Read dimensions and attributes of data (*read_nc_boeing*).
   Close netCDF file.
   Go to step 9 below.
6. Allocate memory for raw data (*getmem32*).
7. Open and read netCDF file (*get_nc_file*).
   Read SWTS wind tunnel parameters (*read_nc_swts*).
   Read model and array coordinates (*read_nc_xyz*).
   Read dimensions and attributes of raw time data.
8. Get calibration data:
   Read sensitivity calibration slopes(*get_sensitivity*).
   Get instrumentation amplitude and phase corrections(*get_nc_cal*).
   Get installation corrections (*get_nc_install*).
   Get free-field calibration (*freefield_coef*).
   Get directivity calibration (*get_nc_directivity*).
9. Send signal to control processor (*pvm*).
   Receive task ID of lead parallel process (*pvm*).
10. Send calibration and wind tunnel data to lead process (*send_nc_data*).
11. Send acoustic time data to lead process (*send_time_data*).
12. Write messages if needed.
13. Exit process (*pvmfexit*).

Parallel Processing Program

The Parallel Processing program is run in the lead process and the subordinate parallel processes. These processes can be performed in one or different CPUs. The leader receives the data from the Control program and the Input program. If the number of processes is greater than one, the leader creates the work assignment tables to subdivide the work between all the parallel processes and sends the data to all the subordinate processes. The leader also receives all the input declarations from the control process to be included in the output file. All the parallel processes perform the FFT of the acoustic data, the pre-scanning calibrations, the scanning of the acoustic sources for each assign frequency, and the post-scanning calibrations. The leader accumulates the data, adds the output processor to the virtual machine, and sends the output data to the output process. The leader also sends the end-of-run flag and any error message to the control process.

The main actions in the ‘mapps_proc’ program are:

1. Enroll themselves with the PVM daemon (*pvm*).
2. Receive input parameters (*input*).
   Leader receives data from control processor (*pvm ,controlupk*).
   If number of processors is greater than one:
   -leader sends data to subordinate parallel processes (*pvm*, *controlpak*).
   -subordinates receive data from leader (*pvm*, *controlupk*).
   Leader defines block and frequency assigment tables, and sends them to subordinates processes if needed.
   Subordinates receive assignment tables if needed.
3. Leader receives all other input declarations from control program to be sent to the output program to be included into the output file (*input_lead*).
4. Allocate memory for large arrays (*getmem*)
5. If ‘boeing’ process, go to step 7*.
6. Leader receives raw time acoustic data from input program and sends it to subordinate parallel processors if needed (*distrib*).
   Subordinate parallel processors receive raw time acoustic data (*getdata*).
7. Perform main calibration and scanning processing (*DoIt*).
   Initialize block health matrix, and set band edge and flat spots flags in health matrix (*calt*).
   Do windowing and FFT (*dofft*).
   Do sensitivity calibration of amplitude and phase (*calf3*).
   Leader accumulates sum of FFT data from all processes, processes *caldb*, and returns data to all subordinate processes (*globsum*).
   Leader gets total power and average power for each channel, sets delta decibels and bad channel flags in health matrix, defines output block health matrix and block and channel array flags, and adds gains to FFT data if required (*caldb*)
   Get cross-correlation matrix of each scanning frequency (*xcor_new*).
   Get global average of scan power (*globavg*).
   *For boeing data (if data_source is ‘boeing’)
      Allocate memory for cross-correlation matrix (*getmem32*)
      Get cross-correlation matrix and input data (*get_rcr_boeing*).

   Change model coordinates to wind tunnel coordinates (*o2wt*).
   Store diagonals of matrix, and accumulate individual spectrum arrays and average (*spectrum*).
   Modify diagonals in matrix if it is required (*diagf*).
   Perform planar (*scanp*) or spherical (*scans*) wave scanning.
   Perform density attenuation calibration correction (*density_cor*).
   Perform directivity calibration if it is required (*directivity_cal*).
   Get maximum and average scan power, and restore diagonals into cross-corrrelation matrix (*scanrcr*).
8. Leader adds output process to virtual machine (*openout*).
9. Leader sends input declarations, scan geometry, and health matrix to output process (*sendout*).
10. Subordinates processes send scan results to leader and leader sends scan results to output process(*scanout*).

11. Set a PVM barrier to all parallel processes (*pvm*).
12. Leader sends end-of-run flag to control process if run is interactive, or sends error message if any error has been detected (*pvm*).
13. Parallel processes exit PVM (*pvmfexit*).

Output Program

The Output program accepts the results from the lead parallel process and combines the results with the setup and raw data information for output to a netCDF file. This process must have access to the file created by the Input program and be run on a processor that has access to the output data storage disk. The main actions in the 'mapps_output' program are:

1. Enrolls itself with the PVM daemon (*pvm*), and gets task identification number of lead process (*pvm*).
2. Receives messages from lead process (*pvm*).
3. Initializes arrays or allocate memory to arrays (*getmem*).
4. Receives general data, scan geometry, and microphone health matrix (*getnml*).
5. Receives scan power data (*getscan*).
6. Calls C subroutine to write netCDF output file (*write_out_nc*).
   Writes netCDF output file (*out_nc*).
   Duplicates netCDF header file (*ncdup*).
   Writes scan power results into netCDF output file (*ncpro12*).
7. Sends end-of-run flag to control process, if run is interactive.
8. Exits PVM process (*pvm*).
9. Stop all PVM processes if error is detected (*pvm*).

Included Subroutines

The included subroutines are common files used in the Fortran or C routines. There are general files used by all the programs, and particular files used by singular programs. The general files are 'parameter.h' and 'fortcall.h'. The singular files are 'ncdup.c', 'ncpro.c', 'corrections.c', and 'fortranarray.c'. There are two other general files, 'fpvm3.h' and 'netcdf.inc', that may be stored together with all the included files or may be stored more conveniently in general locations together with the PVM and netCDF software, respectively.

The main parameters of the MAPPS programs are defined in the parameter.h file. All these parameters have been defined as integer*4 data type and are:

'mproc' - the maximum number of parallel processors
'mrows' - the maximum number of samples in a data block for each microphone
'mmic' - the maximum number of microphones, plus one for the reference channel
'mblks' - the maximum number of raw data blocks
'mfrq' - the maximum number of frequencies to be processed
'mscan' - the maximum number of scanned points to be processed
'mneu' - the maximum number of neuron points for interpolation in calibration

The ‘fortcall.h’ file contains information required to correctly transfer parameter lists between Fortran and C.

The ‘ncdup.c’ file duplicates the raw data netCDF file and sets the unlimited dimension as one. It is included in *get_nc* of the Input program.

The ‘ncpro.c’ file writes the processing setup parameters and processing results to the netCDF output file. It is included in *out_nc* of the Output program.

The ‘corrections.c’ file defines the free-field calibration corrections for 1/4 inch microphones, B&K model 4135, and GRAS model TMS140BF. It is included in the *freefield_coef* of the Input program. Corrections for additional models should be included in this file. This file also applies the general neural net fit correction used in *get_nncor* of the Input program for calibration corrections.

The ‘fortranarray.c’ file provides the Fortran/C conversion of array elements. It is included in *out_nc* of the Output program.

The PVM file ‘fpvm3.h’ defines all the Fortran definitions of the parallel virtual machine.

The netCDF file ‘netcdf.inc’ defines the Fortran interface for the netCDF calls.

Running in Batch Mode

The parallel processing can be run in batch mode to allow data processing without the requirement for an operator to be present. This is accomplished through a shell script. Each run and point number must have a unique settings file that can be created by using a text editor to modify a master settings file. Once these settings files are created, they can be run using a script similar to that shown below. PVM must have been activated by calling the pvm command and using the quit command from the terminal window before the script can be run. This method meets the requirement to be able to process one day’s runs for examination the next day.

```
echo ‘quit | pvm’
echo ‘105000PMA00014003cntrl10’
$PVM_ROOT/bin/$PVM_ARCH/mapps_con < 105000PMA00014003cntrl10
echo ‘halt | pvm’
echo ‘quit | pvm’
echo ‘105000PMA00014004cntrl10’
$PVM_ROOT/bin/$PVM_ARCH/mapps_con < 105000PMA00014004cntrl10
echo ‘halt | pvm’
```

The mapps_con is the executable of the MAPPS Control program. The settings files ‘105000PMA00014003cntrl10’ and ‘105000PMA00014004cntrl10’ are created from the settings file generated from the Processing Control Interface.

Visualization Software

Efficient and versatile visualization of array processed results is an essential part of MAPPS. Display of array results presents a complex problem for the system developer. Many instrumentation systems produce easily understood and presented two- or three-dimensional data sets; however, array processing results in five-dimensional data sets. The five dimensions consist of the physical x, y, and z scan geometry as well as frequency and amplitude. The size and complexity of the results puts a larger burden on the visualization software than in the past where the majority of effort was placed on the data reduction. A commercial off-the-shelf graphics program was used as the foundation to reduce the effort required for developing this complex visualization software. PV-WAVE from Visual Numerics (http://www.vni.com) was chosen as the graphics foundation software for its ability to handle large complex data sets. A custom user interface called Mview was written in PV-WAVE CL, a fourth generation language for data visualization. Mview was written specifically to view data processed with MAPPS and is designed to be easily extensible.

Main Module

The Main Module is created when Mview is launched and is shown in figure 19. This module is used to load in the netCDF data sets for viewing. Multiple data sets can be loaded for comparison with the name of each data set displayed in the text window of the Main Module. Once a data set is loaded, the Overview Module will automatically be started. The Main Module is also used to launch functions that affect the entire application, such as loading in a new color table. The menu choices for the Main Module are:

File->Load:	Load a data file in the MAPPS netCDF format for display.
File->Quit:	Quit the application.
View->Source Integration:	Start the source integration Module. This is used to load and display the results of source integration processing.
Options->Load Color Table:	Select a new color table that is applied to all windows.
Options->Animation Speed:	Set the animation speed for all of the data sets to fast, medium, or slow.
Options->Animation->Start ALL:	Start the animation for ALL of the loaded data sets.
Help->Help:	Display the Main Module Help file.

Overview Module

The Overview Module is launched when a data set is loaded in the Main Module and is shown in figure 20. The Overview Module gives an overview of all the frequencies in this run/point and allows the user to select which frequency is shown in other child modules. There are three values for each frequency plotted in this window: Max, Avg, and Asp. Max is the maximum decibel value in the scan surface independent of position for each frequency. Avg is the average decibel value of the noise map for the scan surface for each frequency. The Asp line shows the average decibel value for all the good microphones before scanning and is representative of a single microphone in the flow. The values of the three curves and the current frequency are shown along the bottom of the window. The combination of curves gives the user feedback on the level of the signals versus the background noise and the dynamic range of the noise map for only those frequencies scanned. The selected frequency is shown with a vertical line in the Overview window and can be changed in several ways. The slider bar can be used to click and drag to a new frequency. The buttons at the bottom of the screen can also be used to change frequencies one at a time. In addition, the “Start Animation” button can be pressed to automatically march through all the frequencies starting at the current frequency. Press the animation button a second time to stop the animation. Closing the Overview Module deletes the corresponding data set from memory.

Various modules for looking at the data can be launched from the menus in this module. Changing the frequency in this window automatically updates the frequency displayed in other child modules (e.g., Imager or Surf Modules). The menu choices for the Overview Module are:

File->Quit: Quit the Overview Module and any modules it has open and delete this data set from memory.

View 2D->Imager: Start the Imager Module which projects the scan surface to a 2-D flat surface.

View 2D->Individual Microphone Health: Start a window that indicates the status of all microphones and blocks during processing.

View 2D->Combined Microphone Health: Start a window that indicates which FFT blocks were actually used in processing after column and row rejection.

View 3D->Surf: Start the Surf Module. This module creates a 3-D surface where elevation corresponds to dB level.

Help->Help: Display the Overview Module Help file.

Imager Module

The Imager Module is the primary means of data visualization in Mview. The Imager Module projects the three-dimensional scan surface onto a two-dimensional plane for the frequency indicated in the Overview Module. The window, shown in figure 21, is broken into four areas with the data displayed on the right. Left clicking at any point on the image in this area will display the decibel value of the nearest point on the scan surface.

Along the left of the window are three areas for controlling the display of data and inputs to the various tools available. The Color Scale area controls the ranges used in displaying the data. There are five schemes for setting the color scale that are chosen with the pop up menu.

Abs Min-Max: Absolute Minimum and Maximum: The upper limit is set to the maximum value of the green Max line in the Overview window for all of the frequencies. The lower limit is set to the minimum of the Avg blue line in the Overview window for all the frequencies.

Rel Min-Max: Relative Minimum and Maximum: The upper limit is set to the maximum scanned surface value for the currently displayed frequency. The lower limit is set to the Avg value for the current frequency.

Abs dB Range: Absolute Decibel Range: The upper limit is set to the maximum value of the green Max line in the Overview window for all of the frequencies. The lower limit is determined by subtracting the value in the “Range” text box from the upper limit.

Rel dB Range: Relative Decibel Range: The upper limit is set to the maximum scanned surface value for the currently displayed frequency. The lower limit is determined by subtracting the value in the “Range” text box from the upper limit.

Manual: The upper limit, lower limit, and range are all set manually by the user.

The data can also be displayed using contour lines either superimposed on the color image or displayed independently. Contour lines step from the lower limit to the upper limit as set by the color scale options. The step value or the number of lines between these limits can be set from the Contours section of the Imager window.

The lower left section of the Imager window is used for command specific information.

The menu choices for the Imager Module control the way the data is displayed and launch additional tools. The File menu allows the user to load a file of points defining the shape of the model. These points are superimposed on the data and displayed in a user selected color. The File menu also allows the user to save the image as a postscript or encapsulated postscript file and to quit the Imager Module.

The View menu controls how the data is displayed in the Imager window. Image and Contours select whether a color graduated pattern and/or contour lines are shown. Legend and Frequency select whether a color scale and/or current frequency value are displayed in the data section of the window. Orientation of the data is controlled through the Flip Image choices. With these choices, the flow direction can be oriented as the researcher desires. Note that for the model projection and the data to coincide they must both be in model coordinates and inches. The color and type of symbol used for the model projection points are selected using the Model Color and Model Symbol menu choices.

Additional tools that can be used to examine or analyze the scan data in further detail are selected under the Tools menu. The current choices under this menu are Profile and Source Integration. The Profile command is used to view the amplitude values along a straight line drawn in the Image window. The line is drawn by clicking the left button at the desired start point and dragging to the desired end point. The starting and ending coordinates are displayed in the command specific area of the Image Window as shown in figure 22 and can be altered here. The scan values along this line are shown in the Profiles window (fig. 23). The x-axis is the delta grid point number from the start of the line to the end. Different line styles, symbols, and grid configurations can be chosen from the menus in this window. This feature is useful in viewing relative shapes and amplitudes of structures of a scan surface. The profile window may be dismissed by choosing “close” under the window manager close function.

The Source Integration menu choice has three options: load points, define points, and load multiple source points. The first option is to load in a file defining the source integration region with a fixed region for all frequencies. This region can also be defined using the Define Points submenu. The user uses the mouse to define a source integration region under this option. The Imager window for this mode is shown in figure 24. The user defines the region by left clicking points on the Imager window to draw a “box” around the area of interest. This box can have up to 99 vertices. Middle clicking stops drawing points and connects the first and last points. The region thus defined may be saved to a file by clicking the “Save Pts...” button in the lower left corner of the Imager window when in Source Integration mode.

The method of defining an integration region contained in Mview is limited to a constant single region that is the same for all frequencies. However, the integration code can have several frequency varying regions defined for integration. These regions must be defined in an external code but can be viewed in Mview by loading the definition file with the Load Multi Source Pts submenu. An example of this feature active is shown in figure 25.

Individual Microphone Health Module

This module displays the status information of the microphones and blocks used in processing. Each row corresponds to one microphone and each column corresponds to an FFT block as shown in figure 26. Each rectangle is color coded to indicate whether the data is good (green) or bad (not green). The color indicates which test was used to discard that block of data from processing. Note that if an FFT block of data is flagged as bad, that block of data is excluded from processing for all microphones. The exception to this is if several FFT blocks are flagged as bad for one microphone, then that microphone is discarded and not that FFT data block. This can be seen in the next module, Combined Microphone Health. Left clicking the mouse on a displayed rectangle will display the Microphone and Block numbers currently under the pointer.

Combined Microphone Health Module

This module indicates which data are actually processed after the elimination of bad block columns and bad microphone rows. This window, shown in figure 27, is an indication of which data blocks and microphones were actually processed. While the Individual Microphone Health Module gives a good indication of the health of individual microphones and blocks, the Combined Microphone Health Module gives a good indication of the amount and quality of the overall data processed. Its format is similar to that of the Individual Microphone Health Module.

Surf Module

The Surf Module (fig. 28) is activated by selecting the Surf option in the Overview Module’s View3D menu. The Surf Module takes the scan surface and displays it as a three-dimensional elevation plot, where elevation corresponds to decibel level. This feature is useful to understand the relative difference between decibel peaks in the scan surface. A file of points defining a model can also be loaded in and displayed beneath the surface plot. The color scale and contour options are the same as those of the Imager Module. Two slider controls let the user rotate the surface plot around the viewing x- and z-axes. Options in the View menu are the same as for the Imager Module except for the lack of flip image choices, which are meaningless for this module.

Source Integration Module

The Source Integration Module is activated by selecting Source Integration from the Main Module's View menu. The Source Integration Module is used to display the results of source integrations. Up to six integration files may be loaded at the same time. This allows the comparison of the integration results from different conditions or, as shown in figure 29, the comparison of different integration regions within the same condition. Moving the slider bar changes the frequency and displays the decibel value of each data set at that frequency. The scale can be either set manually or determined automatically. The two types of values which can be viewed are the source integration values and the integration area maximum values. The source integration program is explained in a following section.

FILE FORMATS

Data longevity was considered in choosing file formats. Historically, data stored as raw binary streams cannot be easily used after a time because of the inherently hidden nature of the data field formats. Additionally, binary files written on one CPU operating system type and brand are not guaranteed to be easily readable on other types of machines. This poses a problem as raw and processed data can "sit in the can" for many years and then be required for research. During this time personnel and computer systems may have changed significantly. Plain ASCII files can be used as an easily readable format. However, text files are very large when dealing with the large data sets generated in array measurements and are thus not acceptable for primary data storage. Careful consideration was given to this problem and a universally accepted self-describing binary file format called Network Common Data Format (netCDF) was chosen as the format for data storage. NetCDF was developed by University Cooperation for Atmospheric Research under a National Science Foundation-sponsored program. NetCDF data is stored in a self-describing, machine-independent data set that can be accessed from many types of platforms. For more information on the netCDF format and supporting software drivers please refer to the Unidata web site located at http://www.unidata.ucar/. The netCDF format allows growth of the format for the data files to include parameters not considered in this version, as long as the existing parameters are not changed to ensure backward compatibility. This is possible because netCDF accesses the stored data by variable name, variable attributes, and index number independent of their location in the file. Thus additions to the data files need only add different variable names or attributes as long as the minimum required variables and attributes are present.

File Name Convention

The file naming convention for MAPPS files conforms to the DARWIN (ref. 3) standard convention for the raw and processed data files. This allows integration of data into the DARWIN system for easy access and integration of results with other test suites.

The file name format for the raw time history, processed, and supporting files is:

ttttttPMArrrrrpppfffffvv.eeee

where

tttttt Six digit test number with leading zero padding
PMA Literal indicating data from the PMAT system
rrrrr Five digit run number with leading zero padding
ppp Three digit sequence or point number with leading zero padding
fffff Five character alpha numeric file type descriptor:
- procd -processed data from the MAPPS processing software, netCDF format
- rawth -Digitized data, netCDF or binary format
- sgout -Scan geometry file, netCDF format
- cntrl -processing control setup file, ASCII text format

vv Two digit version number used in tracking.
eeee Variable length file type extension:
- nc -netCDF format
- bin -binary data
- proj -model projection file (ASCII text)
- txt -general ASCII text

All fields except the extension (eeee) must be present and of the prescribed length. They must be fully populated by using leading zero padding.

Raw Time History Data File (ttttttPMArrrrrppprawthvv.nc)

This file contains the digitized raw time history data in counts obtained from the acquisition system as well as the wind tunnel parameter data. In addition, the data file includes tracking information for each channel's set of instrumentation. The tracking information includes the manufacturer, model number, and serial number for each part of the channel's instrumentation. This allows calibration information for subassemblies to be applied as well as the ability to be able to recreate the test setup. The raw time history data file also contains information describing all of the coordinate systems. The version number (vv) used in this file's header is the designator for the version of the raw data file.

A detailed explanation for each variable and field in the raw data netCDF formatted file is included in Appendix 5.

Processed Data File (ttttttPMArrrrrpppprocdvv.nc)

The processed results are stored in this binary netCDF formatted file. The processed information is appended to the header information from the raw time history netCDF file with the dataar1 variable time dimension set to 1. This gives full tracking of the processed results. The version number (vv) used in this file is the processing version number. The tracking back to the specific raw data file is specified in the global attributes section. All of the processing setup information is included in this file. See Appendix 6 for a detailed explanation of the format of this file.

Control Settings File (ttttttPMArrrrrpppcntrlvv)

The settings file is used to store all the settings from the Process Control Interface and transfer them to the processing software. The settings file is an ASCII file that is formatted as a Fortran namelist of inputs. The variables contained in the settings file are explained in Appendix 6 under the procsettup variable. The attributes of the procsettup variable match the fields contained in the settings file. An example of the settings file is shown in Appendix 7.

Test Model Geometry File

This Plot3d format binary grid file describes the model used in the acoustic test. This file may be single- or multi-zone, and should approximate the shape of the actual test model. By using this file as input to ‘findprojection’, a file of points will be generated to overlay with the data set shown in Mview.

Projected Model File

This file is the output of the ‘findprojection’ program that contains the image of the model projected onto the scanning surface. This file must be an ASCII file in column format, where the first column contains the x-coordinate, the second column contains the y-coordinate, and the third column contains the z-coordinate. The points in this file are in inches in the model coordinate system.

Level Integration Files

There are three ASCII files associated with level integration. The first is the area definition input file. The first line of this file is the number of frequencies to be processed. The second line is a switch, with ‘1’ indicating that each frequency requested must have its own set of integration regions specified. The frequency based area definitions follow, with one definition required for each frequency to be integrated. The first line of the frequency based area definitions contains the frequency number (starting at zero) and the frequency value in hertz. The next line has the number of integration areas for this frequency followed by the number of vertices for each area. For example, the line containing “3 6 7 8” would mean three integration areas having six, seven, and eight vertices, respectively. The x, y, and z coordinates for each vertex follow with one vertex entered per line. In other words, for the example given, there would be 21 vertex lines with the first 6 lines being for the first area, next 7 for the second area, and last 8 for the third area. If no integration regions exist for a requested frequency then a ‘0’ can be entered for the number of integration regions.

If the switch in the second line is set to ‘0’ then the integration areas used for the first frequency will be used for all frequencies. In this mode the first frequency definition area must contain a full area definition. However, the rest of the frequency definition areas just contain one line each. That line contains the frequency number and value to be processed.

The output from the level integration are two ASCII files that contain the integrated spectra. The first file contains the integrated values for the frequencies requested from the area definition file. Also listed for the frequencies requested are the maximum scan values contained in the defined integration regions.

Five metrics are listed in this file that combine the frequency values, one Overall Integrated Level and four Perceived Noise Levels (PNL). The Overall Integrated Level is the power summed combination of all the narrow band levels. The Perceived Noise Levels are generated using two methods, with model and full scale values for each method. The first PNL method is the classic method that mimics the analog filter roll off shapes in the combining of frequency bins. The implementation of this method is based on code obtained from Lewis Research Center and is thus labeled the Lewis method. The second method assumes rectangular shapes for the frequency combining bands that has the effect of no energy loss from filter shaping. The second method is labeled the Energy Conservation method. The two methods will give similar results for fully populated and narrow frequency resolution data. Each method is also applied to the integrated spectra after converting the integrated narrow band data to full scale using the model scale value contained in the processed netCDF file.

The second output file contains the third octave values for the narrow band spectrum resulting from the source integration. This file contains the same metrics as the narrow band integration results file with the addition of a tone corrected Perceived Noise Level for model and full scale. The number of narrow band frequency bins contained in each one-third octave must be at least 10 or the third octave will be set to zero.

Calibration Files Used by the Processing Program

Several calibration files are used by the processing code to obtain calibrated results. The process for obtaining these files is complex and is shown in figure 30.

Calibration Sensitivity File

The Calibration Sensitivity file is an ASCII file that contains the Pascals per volt calibration for each channel. Each line contains the conversion for one channel. Thus, if there are 100 channels, then there are 100 lines of values. A ‘1.0’ should be entered for channels that do not contain microphones.

Instrumentation Calibration File

The Instrumentation Calibration File is a netCDF file that contains the information about each instrument calibration and all the parameters that define the neural net model of each instrument. Neural net models representing the average behavior of each instrumentation model may also be stored here. The attributes for the variables “microphone”, “preamp”, “powersupply”, “wiring”, and “filter” in the netCDF file describe the neural net models for each of these instruments and how the neural net model parameters are stored. There is one variable for each piece of instrumentation and/or instrumentation model. For an individual instrument, the variable name encodes the type of instrument and its serial number. For representative instrument models, the variable name encodes the type of instrument, the fact that it is a representative model, and its model number. For example, the variable “micsn1518106” is for the microphone with serial number 151806 and “micsnrepTMS140BF” is for the representation of microphones with model number TMS140BF. The attributes for each instrument or representative instrument variable contain information about that instrument, its calibration, and the parameters of the neural net model. The Instrumentation Calibration File may be combined with other calibration netCDF files.

Installation Calibration File

The Installation Calibration File is a netCDF file that contains information about the installation effect at each microphone location in an array. The attributes for the variable “installation” contain a description of the neural net model, instructions for evaluating the model, and descriptions of how the parameters are stored in this file. There is one variable for each microphone location. The variable name encodes the information that this is an installation calibration model and an index number for the microphone location. For example, the variable “instalmic1” identifies this as an installation calibration for location number 1. The number of the location has no intrinsic meaning, it is only the order of the microphone in the Intermediate Speaker Array Transfer Function File. The attributes of the variable store information about the array, the microphone location, and parameters needed to evaluate the neural net model. The Installation Calibration File may be combined with other netCDF files for other types of calibration; however, due to the naming conventions only one array installation model may be stored in any one netCDF file.

Directivity Calibration File

The Directivity Calibration File is a netCDF file that contains information about the directivity calibration of the array and the neural net model of the directivity correction. The attributes for the variable “speaker” describe the neural net model, how to evaluate the model, and how the parameters are stored in this file. There is one variable for each array. The variable name encodes the identification of this as a directivity calibration and the name of the array. For example, the variable “dir710-01-100” is for the directivity of array 710-01-100. The attributes of the variable store the name of the array, the date the neural net model was computed, and all the parameters needed to evaluate the neural net model of the directivity. The Directivity Calibration File may be combined with other netCDF files.

Intermediate Calibration Files

Some calibration files are not used by the processing code but are needed by various utility programs for the calibration process and are described in this section.

Intermediate Instrumentation Calibration File

The transfer function information for instrumentation is stored in an ASCII file. The first 12 lines are header information, the 13th and 14th lines are reserved for additional header information to be added later, the 15th line lists the columns of data and the 16th line through the end of the file contain the data in four columns. The four columns are frequency index (count), frequency in hertz, magnitude in decibels, and phase in degrees. The example below shows the header for a microphone diaphragm calibration. Since there was no institutional wiring or filter contained in this calibration, these fields are marked with an X.

```
Diaphragm Calibration
Sunday, February 1, 1998
Pressure   14.7 psia
```

```
Input Signal:   Sine Sweep
Temperature:   24.0  °C
Calibration  frequency  247.3  Hz  Level  144  dB  Voltage  0.98  V
Institutional Wiring Channel   X
Signal Conditioning   20 dB
Diaphragm   SN 7188  brand  GRAS  model  TMS140BF  size 1/4"
Preamp  SN  7311  brand GRAS  model  TMS126AC  size 1/4"
Power Supply  SN  7500.A  brand  GRAS  model  TMS112AA
Filter SN  XXXXXXX brand XXXXXX model  XXXXXX
blank line 1
blank line 2
COUNT   HZ      MAG     PHASE
0.000000E+0     1.000000E+2     -4.272347E+1    -1.786992E+2
1.000000E+0     2.248750E+2     -4.272871E+1    -1.795871E+2
...
```

Intermediate Speaker Free-Field Transfer Function File

The speaker free-field transfer function file is in ASCII format and contains the transfer functions from the reference channel to a free-field microphone. These free-field microphone measurements are made at many locations to obtain the speaker characteristics over the region of the physical array microphones. This file is produced by the program ‘spdefxferfun’ and is used in the MATLAB script ‘MakeInstalModel’. The first five rows contain header information; row 1 contains 1000*run number + point number, row 2 contains alpha in degrees, row 3 contains beta in degrees, row 4 contains radius in inches, and row 5 contains temperature in degrees Fahrenheit. Starting with the sixth row, the first column contains frequency in hertz and all the other columns contain the transfer function for each measurement location. An example is shown below:

```
0       runpt   runpt   ...
0       alpha   alpha   ...     (deg)
0       beta    beta    ...     (deg)
0       rad     rad     ...     (inches)
0       T       T       ...     (deg F)
f       TFun    TFun    ...
.       .       .       ...
.       .       .       ...
```

Intermediate Speaker Definition File

The Intermediate Speaker Definition File is a netCDF file that contains information about the speaker and a neural net model of the speaker transfer function. The attributes for the variable “speakerdefinition” contain a description of the neural net model, instructions for evaluating the model and descriptions of how the parameters are stored in this file. There is one variable for the speaker. The variable name encodes the information that this is a speaker definition model and a name of the speaker. For example, the variable “speaker SJ1” identifies this as a speaker definition for the speaker

SJ1. The attributes of the variable store information about the speaker and parameters needed to evaluate the neural net model.

Intermediate Speaker Array Transfer Function File

The speaker array transfer function file is an ASCII format file that contains transfer functions from the reference channel to the individual array microphones. This file is produced by the program ‘sparrayxferfun’ and is the input into the MATLAB script ‘MakeInstalModel’. The first ten rows contain header information: the first three rows contain the x, y, and z coordinates of the microphones in the array coordinate system in inches; rows 4 through 6 contain the radius, theta, and phi values for the microphones in the speaker spherical coordinate system; row 7 contains the FFT block size; row 8 contains the number of FFT blocks averaged; and row 9 contains the temperature in degrees Fahrenheit. The tenth row contains the xducertype indicator from the raw data file. If this type is set to ‘1’ then it is an array microphone. Starting with the eleventh row, the first column contains the frequency in hertz and all the other columns contain the transfer function. An example is shown below:

```
0       x       x       ...     (inches)
0       y       y       ...     (inches)
0       z       z       ...     (inches)
0       rad     rad     ...     (inches)
0       theta   theta   ...     (degrees)
0       phi     phi     ...     (degrees)
0       bs      bs      ...
0       nb      nb      ...
0       T       T       ...     (deg F)
0       1       4       ...     (xducertype)
f       TFun    TFun    ...
.       .       .       ...
.       .       .       ...
```

Intermediate Directivity Calibration Files

The intermediate directivity calibration files are ASCII format files that contain spectra from all of the scans made for the directivity calibration. These files are produced by the program ‘dirprennet’ and used in the MATLAB script ‘MakeDirFile’. One file contains the amplitude spectra of the maximum scan value, one file contains the amplitude spectra of the average of all the good microphones, and one file contains the amplitude spectra of the reference signal. In each file, the first five rows contain header information: row 1 contains the run point designator calculated using 1000*run number + point number, row 2 contains the angle alpha in degrees, row 3 contains the angle beta in degrees, row 4 contains the radius in inches, and row 5 contains the temperature in degrees Fahrenheit. Starting with the sixth row, the first column contains frequency in hertz and all the other columns contain amplitude spectra in decibels. An example is shown below:

```
0       runpt   runpt   ...
0       alpha   alpha   ...     (deg)
0       beta    beta    ...     (deg)
```

```
0       rad     rad     ...     (inches)
0       T       T       ...     (deg F)
f       Amp     Amp     ...     (dB)
.       .       .       ...
.       .       .       ...
```

UTILITY PROGRAMS

While the basis of MAPPS is the three main elements consisting of processing control interface, parallel processing software, and visualization software, there are several codes that perform supporting tasks. These supporting codes perform such functions as finding the model projection, performing source integration, determining calibration curve fits, and converting formats.

Find Model Projection

An accurate depiction of the model component locations with respect to the scanning surface is required to understand the source locations. Because the scan surface can be an arbitrary surface in space and may be far from the model components, it is insufficient to simply overlay the model and scan surface. What is seen by the array is equivalent to placing an arbitrary screen between your eyes and something in space. What you see on the screen is the intersection of rays from that object to your eyes. Thus the model component points must be projected onto the arbitrary scan surface to get an accurate depiction of source locations. This concept is illustrated in figure 31. The C program ‘findprojection’ was written to perform this function. It takes as input the scan surface geometry and microphone array locations contained in the processed netCDF file and the model geometry file. The output is a file containing the projected points, which is loaded into Mview and overlaid onto the scan results. The input model geometry and the scan surface are in inches in the model coordinate system and the microphone array is in inches in the wind tunnel coordinate system. The model geometry and scan points are converted to wind tunnel coordinate system internally and then back to model coordinate system upon output of the projected points. It is essential that the model geometry points must be in the model coordinate system and the rotation matrix and translation vector from model to wind tunnel coordinate systems must be correct in the netCDF file.

Source Level Integration

Array processing yields the correct result in the ideal case of scanning through one point source in an infinite domain. The array response from scanning the point source gives the squared acoustic pressure at the center of the array due to that ideal source. If the source is a distributed source, or multiple sources (including reflections), the array response may not give a good measure of the total acoustic pressure. If sources are widely separated, each source should produce little effect on the array response of the others. When sources are close to each other, the array response for one will be affected by the other and they may even appear as merged.

Robert Dougherty (personal communication, The Boeing Company, October 17,1995) suggested a method to evaluate the effect of multiple or distributed sources within an area from the array response map. A region is defined in the scan area. The integral of the array response over that region is computed and referenced to the integral over the same region of the array response to an ideal point source. This new metric relates the source strength to the integral of the array response instead of the maximum of the array response. This ratio is then applied to the maximum source value in the integration area. The C routine ‘levelint’ was developed which implements this scheme.

‘Levelint’ can integrate multiple areas that are different for each frequency. The final integrated value is obtained by adding the results of the individual area integration at each frequency. ‘Levelint’ takes as input an ASCII file that defines the integration areas as a function of frequency. The inputs are:

1st line: processed netCDF data file location including full path
2nd line: ASCII input file which contains the area definitions including full path
3rd line: ASCII output file name including full path
4th line: comment line to include in output file
5th line: delta dB down from maximum in the integration area to include in integration
6th line: conversion factor to convert units contained in processed netCDF data file to meters
7th line: switch to output third octave file (0 for no, 1 for yes)
8th line: switch to process more cases (0 for no, 1 for yes)

If the switch for third octave output file is turned on, then a file with the same name as that in the 3rd line will be output but with a “to” appended. If the switch for more cases is set to one then lines 1 through 8 are repeated until the switch is set to zero, at which point the program will exit.

Processed NetCDF to Plot 3D Conversion

The scan surface geometry and data values are stored in the netCDF file in variables that relate directly to Plot3d format. This format makes it easy to extract the data sets and read them into the Flow Analysis Software Toolkit (FAST).

‘Netfast’ is a utility which extracts the sgout scan surface and function file data from a netCDF data set, and writes binary Plot3d scan surface and function files. Note that the grid file output by ‘netfast’ is one layer of the actual scan surface and must be expanded to have one layer for each frequency contained in the function file. To do this, run the utility ‘grexpand’.

Expanded Grid File Generation for FAST

The ‘grexpand’ routine reads a single plane of scan grid data and duplicates the grid for each frequency for use in the FAST visualization environment. The number of frequencies is read from the scan pressure function file generated by ‘netfast’.

Usage: grexpand gridfile pressurefile output_filename

The gridfile must be a binary plot3d single zone grid file. The pressurefile must be a binary plot3d function file.

Model Coordinate Alignment

The C program ‘movit’ allows the user to apply certain translations and rotations to a model in order to align it to the proper model coordinate system. This program reads in an object from a Plot3d ASCII grid file, applies FAST translation/rotations as entered by the user, and then outputs the object to a Plot3d ASCII grid file. The transformations are modeled after the Transform Typeins dialog in FAST.

Calibration Generation Utilities

There are several steps required to generate the four calibration files required for fully calibrated array results. The chart shown in figure 30 shows the steps required and each program or script is described below.

Speaker Free-Field Definition Transfer Function Generation

The C program ‘spdefxferfun’ generates the transfer function between the free-field microphone and the calibration source signal for all the locations in the free-field measurement matrix. The program first calculates the offset between the calibration and free-field signals via cross correlation to account for the retarded time. The cross spectrum is then calculated between the reference microphone and calibration signal using this offset. The cross spectrum of the microphone and calibration signal and the auto spectrum of the calibration signals are averaged and then the transfer function is calculated. The equation is:

$$T = \frac{\left\langle F_{microphone} \cdot F'_{calibration} \right\rangle}{\left\langle F_{calibration} \cdot F'_{calibration} \right\rangle} \tag{35}$$

All electrostatic, sensitivity, and free-field calibrations are applied to the signals after the FFT is calculated and before the correlations are calculated. The results of these calculations are written to a file which contains the transfer functions for all free-field definition locations. This file is then fed into the MATLAB script ‘MakeSpeakModel’ to generate the free-field definition neural net curve fit.

Speaker Array Transfer Function Generation

The C program ‘sparrayxferfun’ generates the transfer function between the array microphones and the calibration source signal for all the microphones in the array at a reference location. The program first calculates the offset between the calibration and each microphone signal. The cross spectrum is then calculated between the array microphone and calibration signal using this offset. The cross spectrum of the microphone and calibration signal and the auto spectrum of the calibration signals are averaged over the requested number of averages and then the transfer function is calculated. The equation is:

$$T = \frac{\left\langle F_{microphone} \cdot F'_{array} \right\rangle}{\left\langle F_{array} \cdot F'_{array} \right\rangle} \tag{36}$$

All electrostatic and sensitivity calibrations are applied to the signals after the FFT is calculated and before the correlations are calculated. The results of these calculations are written to a file which contains the transfer functions for all array microphones. This file is then fed into the MATLAB script ‘MakeInstalModel’ to generate the installation effects neural net curve fit.

Intermediate Directivity Calibration File Generation

The C program ‘dirprennet’ accumulates directivity information from processed data files and outputs them as tables. Three tables as ASCII files are output: one each for the scan maximum value, scan average value, and the average spectra of all microphones. These files are used by the MATLAB script ‘MakeDirModel’.

Daily Sensitivity Generation

The ‘dailysensitivity’ program calculates a transfer function metric for a reference and a daily speaker measurement. The ratio of these two metrics is used to modify a reference sensitivity file. The metric for each measurement is calculated by first applying the proper gains and electrostatic calibrations. The transfer function between each array microphone and a reference source is then calculated using the same method as the ‘sparrayxferfun’ program. The transfer functions are then power averaged over a user-defined frequency range as well as a user-defined subset of array microphones. The reference sensitivities for all array channels are then multiplied by the ratio of the reference metric to the daily measurement metric to give the corrected sensitivity file to be used by the parallel processing code.

Instrumentation Neural Net Curve Fit

Microphone Diaphragm Curve Fit

The MATLAB script ‘MakeMicModel’ will make neural net models of the amplitude and phase response for the microphone power supply, preamplifier, diaphragm, filter, and wiring. For the calibration of instrumentation connected in a series, the order of neural net modeling is important. During the neural net model generation process, consideration is given to all instruments listed in Intermediate Instrumentation Calibration File. Exactly one piece of instrumentation will be identified with each neural net model. If any other instrumentation are listed in the file, the program will attempt to remove the effect of the other instrument(s). The effect of the other instrument(s) will be removed if a neural net model exists in the identified netCDF Instrumentation Calibration File of that instrument or that instrument model. This feature allows mixing instrumentation without losing calibration. On the other hand, if the user does not want to calibrate each instrument separately, the series can be calibrated as a whole and identified with the microphone diaphragm serial number.

The transfer function of the microphone diaphragm may be modeled as a function of wave number at atmospheric pressure or as a function of wave number and pressure. For a microphone diaphragm, the Individual Instrumentation Calibration File for that diaphragm will contain a calibration at only one pressure. Multiple files are used when the model includes pressure. When multiple files are used, neural net models may also be made of the delta pressure amplitude and phase. The delta pressure is measured from the pressure in the first file:

$$\Delta|P| = |P_n| - |P_1| \tag{37}$$

$$\Delta\angle P = \angle P_n - \angle P_1 \tag{38}$$

For all other instruments, the neural net model is made as a function of frequency. The example below shows the program inputs needed to make amplitude and phase models of the diaphragm stored in the intermediate file “/usr/people/mosher/ARRAY/PRESSCAL/CALRESULTS/DATA/D1” and store the results in the file “/usr/people/mosher/ARRAY/temp3.nc”. The final slash, “/”, must be included when entering the name of any directory. The user inputs are in Courier and the program prompts are in bold Courier font. Enter MATLAB by typing matlab at the UNIX prompt; the prompt inside MATLAB is >>.

```
>> MakeMicModel
Input directory name in quotes for NetCDF (default:
/usr/people/mosher/ARRAY/PRESSCAL/)
Input file name in quotes for NetCDF (default: temp.nc) ‘temp3.nc’
Use 1 file to model a single diaphragm at atmospheric pressure.
Use multiple files to model the delta pressure from the first file.
Use 1 file to model any other instrument.
Input number of files to read (default: 1 ) 1
Input file name ‘/usr/people/mosher/ARRAY/PRESSCAL/CALRESULTS/DATA/D1’

Select the instrument.

Enter 1 for Power Supply
Enter 2 for Preamp
Enter 3 for Diaphragm 3

Warning: file /usr/people/mosher/ARRAY/PRESSCAL/temp3.nc does not exist
> In /usr/people/mosher/ARRAY/PRESSCAL/ReadMicModel.m at line 173
  In /usr/people/mosher/ARRAY/PRESSCAL/CorrectData.m at line 55
  In /usr/people/mosher/ARRAY/PRESSCAL/MakeMicModel.m at line 308

Warning: file /usr/people/mosher/ARRAY/PRESSCAL/temp3.nc does not exist
```

```
> In /usr/people/mosher/ARRAY/PRESSCAL/ReadMicModel.m at line 173
  In /usr/people/mosher/ARRAY/PRESSCAL/CorrectData.m at line 77
  In /usr/people/mosher/ARRAY/PRESSCAL/MakeMicModel.m at line 308

Warning: file /usr/people/mosher/ARRAY/PRESSCAL/temp3.nc does not exist
> In /usr/people/mosher/ARRAY/PRESSCAL/ReadMicModel.m at line 173
  In /usr/people/mosher/ARRAY/PRESSCAL/CorrectData.m at line 118
  In /usr/people/mosher/ARRAY/PRESSCAL/MakeMicModel.m at line 308

Warning: file /usr/people/mosher/ARRAY/PRESSCAL/temp3.nc does not exist
> In /usr/people/mosher/ARRAY/PRESSCAL/ReadMicModel.m at line 173
  In /usr/people/mosher/ARRAY/PRESSCAL/CorrectData.m at line 55
  In /usr/people/mosher/ARRAY/PRESSCAL/MakeMicModel.m at line 308

Warning: file /usr/people/mosher/ARRAY/PRESSCAL/temp3.nc does not exist
> In /usr/people/mosher/ARRAY/PRESSCAL/ReadMicModel.m at line 173
  In /usr/people/mosher/ARRAY/PRESSCAL/CorrectData.m at line 77
  In /usr/people/mosher/ARRAY/PRESSCAL/MakeMicModel.m at line 308

Warning: file /usr/people/mosher/ARRAY/PRESSCAL/temp3.nc does not exist
> In /usr/people/mosher/ARRAY/PRESSCAL/ReadMicModel.m at line 173
  In /usr/people/mosher/ARRAY/PRESSCAL/CorrectData.m at line 118
  In /usr/people/mosher/ARRAY/PRESSCAL/MakeMicModel.m at line 308
Enter a lower frequency if you want the maximum lower than 100000 50000
Enter yes in quotes if you want to model absolute amplitude ‘yes’
Enter yes in quotes if you want to model absolute phase ‘yes’
Input number of neurons in layer 1 (default: 10 ) 9
Input normalized rms goal (default:   0.0075) .01
Input maximum number of epochs to train (default:  30.0000)
TRAINLM: 0/30 epochs, mu = 0.01, SSE = 9978.77.
TRAINLM: 3/30 epochs, mu = 0.01, SSE = 1.08375.

cc =

    1.0000    0.9998
    0.9998    1.0000

TRAINLM: 0/30 epochs, mu = 0.01, SSE = 2.0893e+07.
TRAINLM: 5/30 epochs, mu = 1, SSE = 641.358.
TRAINLM: 6/30 epochs, mu = 1, SSE = 295.389.

cc =

    1.0000    0.9989
    0.9989    1.0000
```

```
TRAINLM: 0/60 epochs, mu = 0.01, SSE = 2.08509e+07.
TRAINLM: 5/60 epochs, mu = 1, SSE = 449.264.
TRAINLM: 6/60 epochs, mu = 1, SSE = 130.8.

cc =

    1.0000    0.9996
    0.9996    1.0000

>>
```

The default values for the number of neurons, rms goal, and number of epochs usually work very well. The script will make three attempts to generate a good curve fit. The curve fit for modeling a diaphragm at atmospheric pressure is considered good if the maximum error in amplitude is within 0.15 dB and the maximum error in phase is within 3 degrees. The curve fit for modeling a diaphragm with delta pressure is considered good if the maximum error in amplitude is within 0.5 dB and the maximum error in phase is within 3 degrees. A good curve fit will not be produced if there is a resonance in the response of the instrumentation within the wave number range of interest. Instrumentation with resonances in this range should not be used in the array. The warning messages occur in this example because no netCDF file was found when the program tried to read neural net models of the power supply and preamplifier. If you want all of the instruments in the series calibrated as a whole, then the warning messages will be given and can be ignored. If you want each instrument calibrated separately, the warning messages will alert you that something is amiss. You can determine the status of all the instruments by reading the resulting netCDF file with the utility program ‘ncdump’.

The neural net curve fits for instrumentation may be run in batch mode. To run multiple cases in batch mode, make an input file for each microphone that looks like the following file, D46.inp:

```
pause(1);
MakeMicModel
‘/usr/people/mosher/ARRAY/PRESSCAL/’            % NetCDF directory
‘calibration19980421.nc’                        % NetCDF file
1                                               % number of pressures
‘/m3d4/soderman/bodata/results/D46’             % input file
3                                               % instrument
100000                                          % maximum frequency
‘yes’                                           % absolute amplitude
‘yes’                                           % absolute phase
                                                % number of neurons
                                                % normalized rms goal
                                                % maximum # epochs
                           %(need 3 blank lines after this for batch)
```

The pause at the top and the three blank lines at the end of the file are needed for the program to run correctly. This input file starts the program and provides all the input to the program. To run in batch, write and run an executable script to run MATLAB with the input files and output files as shown below:

```
matlab < /directory/D1.inp > /directory/D1.out
...
matlab < /directory/D100.inp > /directory/D100.out
```

Viewing Microphone Diaphragm Curve Fit

The MATLAB script 'SummarizeCalQuality' will read the instrumentation calibration netCDF file, "file.nc", and generate a table showing the results. The resulting table is output as the file "file.nc.quality". Any curve fit that did not pass the quality check for maximum error should be examined. The MATLAB script 'PlotMicModel' will plot both the data and neural net curve fit for a power supply, preamplifier, microphone diaphragm, filter, or wiring. Determination to exclude a poor quality instrument can be made after examining this plot.

Representative Microphone Curve Fit

The MATLAB script 'MakeRepMicModel' will make neural net models of the representative amplitude and phase responses of the microphone power supply, preamplifier, diaphragm, filter, and wiring. This representative model is based on the responses of all of the specific instruments in the instrumentation calibration files. Each input file is for one specific instrument. Treatment of the data is essentially the same as it is in making models of individual instruments, except the representative model will approximate the average behavior for all the specific instruments. The example below shows the inputs to make a representative model of power supplies. Two power supplies were used.

>> MakeRepMicModel
Input directory name in quotes for NetCDF (default: /usr/people/mosher/ARRAY/PRESSCAL/)
Input file name in quotes for NetCDF (default: temp.nc) 'temp3.nc'
Input number of files to read (default: 1) 2
Input file name '/usr/people/mosher/ARRAY/PRESSCAL/CALRESULTS/DATA/PS1b'
Input file name '/usr/people/mosher/ARRAY/PRESSCAL/CALRESULTS/DATA/PS41'

Select the instrument.

Enter 1 for Power Supply 1
Enter a lower frequency if you want the maximum lower than 100000
Enter yes in quotes if you want to model absolute amplitude 'yes'
Enter yes in quotes if you want to model absolute phase 'yes'
Input number of neurons in layer 1 (default: 10)
Input normalized rms goal (default: 0.0200)

```
Input maximum number of epochs to train (default:  30.0000)
TRAINLM: 0/30 epochs, mu = 0.01, SSE = 2958.09.
.
.
TRAINLM: 30/30 epochs, mu = 1e-05, SSE = 0.263802.

TRAINLM: Network error did not reach the error goal.
  Further training may be necessary, or try different
  initial weights and biases and/or more hidden neurons.


cc =

    1.0000    0.7830
    0.7830    1.0000

TRAINLM: 0/30 epochs, mu = 0.01, SSE = 37701.7.
TRAINLM: 5/30 epochs, mu = 1, SSE = 40.1887.
TRAINLM: 6/30 epochs, mu = 1, SSE = 15.4508.

cc =

    1.0000    0.9995
    0.9995    1.0000

>>
```

In this example, the amplitude curve fit did not converge and the phase curve fit did converge. The resulting plots must be examined to determine if the curve fits are adequate. In this example the lack of convergence is due to scatter in the amplitude response of the power supplies. The script will generate a plot showing the model and all the data to generate the model. This plot will be saved as a postscript file in the directory identified for the netCDF file. The name of the postscript file will include the name of the netCDF file, type of instrument, and type of model, so it should be readily identifiable.

Speaker Model Neural Net Curve Fit

The MATLAB script ‘MakeSpeakModel’ will make a neural net model of the transfer function of the speaker. Start the curve fit program by typing ‘MakeSpeakModel’ inside MATLAB after running the C program ‘spdefxferfun’. The program has default values for all input parameters that may be changed when the program prompts for input. The frequency range should include only those frequencies where measurements have a good signal to noise ratio and where installation effect will be needed for noise measurements. The reference location is an item strictly internal to the speaker definition and does not need to match other reference locations used in other aspects of calibration. The reference location should be a location where you know there is a good transfer function measurement. The script will select the data point closest to the reference location entered. The first set of default values for the number of neurons, error level, and number of epochs is generally good. These are used to develop a neural net model of the transfer function at the reference location. In the middle of the script, the option is provided to view the transfer functions before selecting the parameters for the main function fitting. Plots will be shown until a frequency of “0” is entered. For data that is nearly axisymetric, enter “2” for the dimension of the function. Most other data requires three dimensions to produce a good curve fit. The next set of parameters for the number of neurons, error level, and number of epochs are suggested starting values. Experience in curve fitting will be needed to select good values. It is best to use as few neurons as possible. If too many neurons are used, overfitting will occur and generate large deviations in the curve fit between data points in the training set. It is best to use one training set with all of the data. Options to train the neural net on subsets of data are provided for cases when the computer is too small for the entire data set. It is recommended that the script to make a speaker model only be run interactively.

```
>> MakeSpeakModel
Input speaker name in quotes (default: SJ1) ‘SJ1’
Input directory name in quotes for speaker transfer function (default:
/usr/people/mosher/ARRAY/SPEAKCAL/EXAMP/SPEAKDEFXFER/)
‘/usr/people/mosher/ARRAY/SPEAKCAL/EXAMP/SPEAKDEFXFER/’
Input file name in quotes for speaker transfer function (default: Tmatrix512) ‘Tmatrix512.v3’
Input directory name in quotes for NetCDF (default:
/usr/people/mosher/ARRAY/SPEAKCAL/) ‘/usr/people/mosher/ARRAY/SPEAKCAL/’
Input file name in quotes for NetCDF (default: temp.nc) ‘temp9.nc’
Minimum frequency = 0 , maximum frequency = 64800
Enter minimum frequency; 2000
Enter maximum frequency: 20000
The default reference location is at alpha = 0 and beta = 0
Input value for alpha (default: 0 ) 0
Input value for beta (default: 0 ) 5
Input number of neurons in layer 1 (default: 15 ) 20
Input normalized rms goal (default:   0.025) .02
Input maximum number of epochs to train (default: 50.0000)50

TRAINLM: 0/50 epochs, mu = 0.001, SSE = 441114.
.
.
.
TRAINLM: 46/50 epochs, mu = 0.0001, SSE = 0.710855.

cc =

    1.0000    0.9972
    0.9972    1.0000

The minimum frequency is 2000 the maximum frequency is 20000
```

```
Enter the frequency you want plotted (default: 1000 , enter 0 to halt) 0
Input number of independent variables (default: 3 )
Enter 1 for wave number only
Enter 2 for wave number and theta
Enter 3 for wave number, theta and phi 3
Input number of neurons in layer 1 (default: 15 ) 12
Input normalized rms goal (default:   0.0350) .05
Input maximum number of epochs to train (default:  200.0000) 100
There are 16744 data points available for training
Maximum number of training sets is 6
Input # of training sets (default: 1) 1
Input number of points in training set (default: 16744)
TRAINLM: 0/100 epochs, mu = 0.1, SSE = 17874.6.
.
.
.
TRAINLM: 100/100 epochs, mu = 0.1, SSE = 411.221.
TRAINLM: Network error did not reach the error goal.
  Further training may be necessary, or try different
  initial weights and biases and/or more hidden neurons.

The minimum frequency is 2000 the maximum frequency is 20000
Enter the frequency you want plotted (default: 1000 , enter 0 to halt)2000
Enter the frequency you want plotted (default: 1000 , enter 0 to halt)0
```

The resulting curve fit should always be viewed to determine if the resultant speaker model is acceptable. Convergence is not a good measure of the goodness of the curve fit. The convergence criteria are a program parameter that may easily be changed by changing the rms goal. One rms goal is not good for all data. If the data contains large scatter, a larger rms goal is appropriate. If the data is very smooth, a smaller rms goal is appropriate.

Installation Neural Net Curve Fit

The MATLAB script ‘MakeInstalModel’ will make a neural net model of the installation effect for each microphone location. The installation effect measures the difference between measurements made with the array microphones and with a well calibrated isolated microphone. Start the curve fit program by typing ‘MakeInstalModel’ inside MATLAB after running the MATLAB script ‘MakeSpeakModel’ and the C program ‘sparrayxferfun’. The program has default values for all input parameters that may be changed when the program prompts for input. The frequency range is determined by the frequency ranges in the files containing the array measurement and the speaker model. The default values for the number of neurons, error level and number of epochs are generally good. The script will attempt to make a curve fit up to three times. The fit is considered good if the maximum error is less than 0.3 dB. The curve fit may be good enough even if these criteria are not met. One curve is generated for each microphone location. The script saves the plot showing each curve fit in a postscript file for later reference.

```
>> MakeInstalModel
Input speaker name in quotes (default: SJ1) ‘SJ1’
Input directory name in quotes for speaker model (default:
/usr/people/mosher/ARRAY/SPEAKCAL/) ‘/usr/people/mosher/ARRAY/SPEAKCAL/’
Input file name in quotes for speaker model (default: temp.nc) ‘temp2.nc’
Input array name in quotes (default: 710-01-100) ‘710-01-100’
Input directory name in quotes for array transfer function (default:
/usr/people/mosher/ARRAY/INSTALCAL/EXAMP/)
‘/usr/people/mosher/ARRAY/INSTALCAL/EXAMP/’
Input file name in quotes for array transfer function (default: Toutest) ‘Toutest’
Input directory name in quotes for output netCDF file (default:
/usr/people/mosher/ARRAY/INSTALCAL/EXAMP/)
‘/usr/people/mosher/ARRAY/INSTALCAL/’
Input file name in quotes for output netCDF file (default: testinstal.nc) ‘temp2.nc’
Input number of neurons (default: 22 ) 22
Input normalized rms goal (default:   0.0200) .02
Input maximum number of epochs to train (default: 50.0000) 50

TRAINLM: 0/100 epochs, mu = 0.001, SSE = 15844.3.
.
.
.
TRAINLM: 100/100 epochs, mu = 0.1, SSE = 5.45016.

TRAINLM: Network error did not reach the error goal.
  Further training may be necessary, or try different
   initial weights and biases and/or more hidden neurons.



cc =

    1.0000    0.9933
    0.9933    1.0000

TRAINLM: 0/100 epochs, mu = 0.001, SSE = 18504.7.
.
.
.
TRAINLM: 100/100 epochs, mu = 0.01, SSE = 8.18653.

TRAINLM: Network error did not reach the error goal.
  Further training may be necessary, or try different
   initial weights and biases and/or more hidden neurons.
```

```
cc =

    1.0000    0.9905
    0.9905    1.0000
```

Directivity Neural Net Curve Fit

The MATLAB script ‘MakeDirModel’ will make a neural net model of the directivity response of the array. The example below shows how to run the program. Start the curve fit program by typing ‘MakeDirModel’ inside MATLAB after running the C program ‘dirprennet’. The program has default values for all input parameters that may be changed when the program prompts for input. In the example below, some parameters were left unchanged and some were changed to show how the program works. Data can be viewed before curve fitting by entering a frequency to view. Data viewing stops when a frequency of ‘0’ is entered. This example converged in 38 epochs. After generating the curve fit, the data and curve fit were viewed at one frequency. The program was halted when a response of ‘0’ was entered for viewing frequency.

```
>> MakeDirModel
Input directory name in quotes for data files (default:
/usr/people/mosher/ARRAY/DIRECTIVITY/)
Input directory name in quotes for NetCDF (default:
/usr/people/mosher/ARRAY/DIRECTIVITY/) ‘/usr/people/mosher/ARRAY/’
Input file name in quotes for average data file (default: diravg710_01_100) ‘diravg’
Input file name in quotes for maximum scan data file (default: dirmax710_01_100) ‘dirmax’
Input file name in quotes for reference channel file (default: dirref710_01_100) ‘dirref’
Input file name in quotes for NetCDF file (default: temp2.nc) ‘fileout.nc’
The default reference location is at alpha = 0 and beta = 0
Input value for alpha (default: 0 )
Input value for beta (default: 0 )
The minimum frequency is 500 the maximum frequency is 20000
Enter the frequency you want plotted (default: 1000 , enter 0 to halt)0
Enter minimum frequency for processing (default 500);
Enter maximum frequency for processing (default 20000); 5000
There are 3420 data points available for training
Maximum number of training sets is 6
Input # of training sets (default: 1) 1
Input number of points in training set (default: 3420)
Enter number of independent variables (2 or 3) (default: 3)
Input number of neurons in layer 1 (default: 15 )7
Input normalized rms goal (default:   0.0100).002
Input maximum number of epochs to train (default: 100.0000)
TRAINLM: 0/100 epochs, mu = 0.001, SSE = 61726.8.
```

```
cc =

    1.0000    1.0000
    1.0000    1.0000

The minimum frequency is 500 the maximum frequency is 20000
Enter the frequency you want plotted (default: 1000 , enter 0 to halt)500
The minimum frequency is 500 the maximum frequency is 20000
Enter the frequency you want plotted (default: 500 , enter 0 to halt)0
>>
```

An option is available to divide the training set into multiple smaller training sets. Generally, it is better to have fewer and larger training sets. However, it is possible to have more data in a training set than the computer can handle. Thus, the number of values in a training set should be tailored to the amount of available memory. Selecting the number of variables, the number of neurons, the convergence criteria, and the number of epochs will take some trials to find good values. You may curve fit to two independent variables, wave number and theta, or three independent variables including phi. The default values for number of neurons, rms goal, and number of epochs are based on experience with a specific data set and may not be appropriate for other data sets. It is best to use the fewest number of neurons that will give a good curve fit. Looking at the data and experience making curve fits are the best guides to selecting the number of neurons to produce a good curve fit. The rms goal may need to be increased for data with a large amount of scatter or decreased for very smooth data.

It is recommended that the script to make the neural net curve fit for the array directivity always be run interactively.

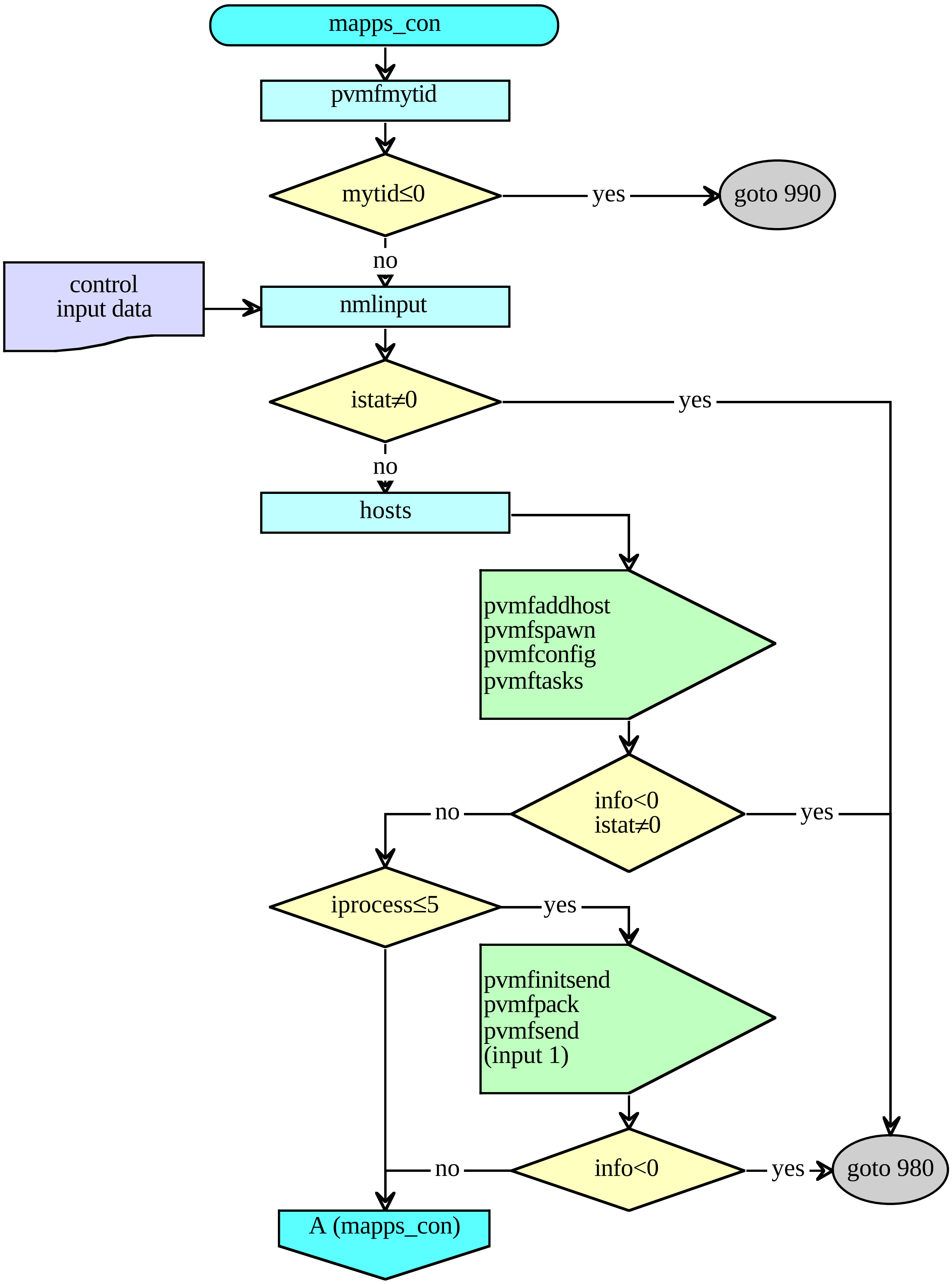
REFERENCES

1. Walatka, Pamela P.; Buning, Pieter G.; Pierce, Larry; and Elson, Patricia: PLOT3D User’s Manual. NASA TM-101067, March 1990.

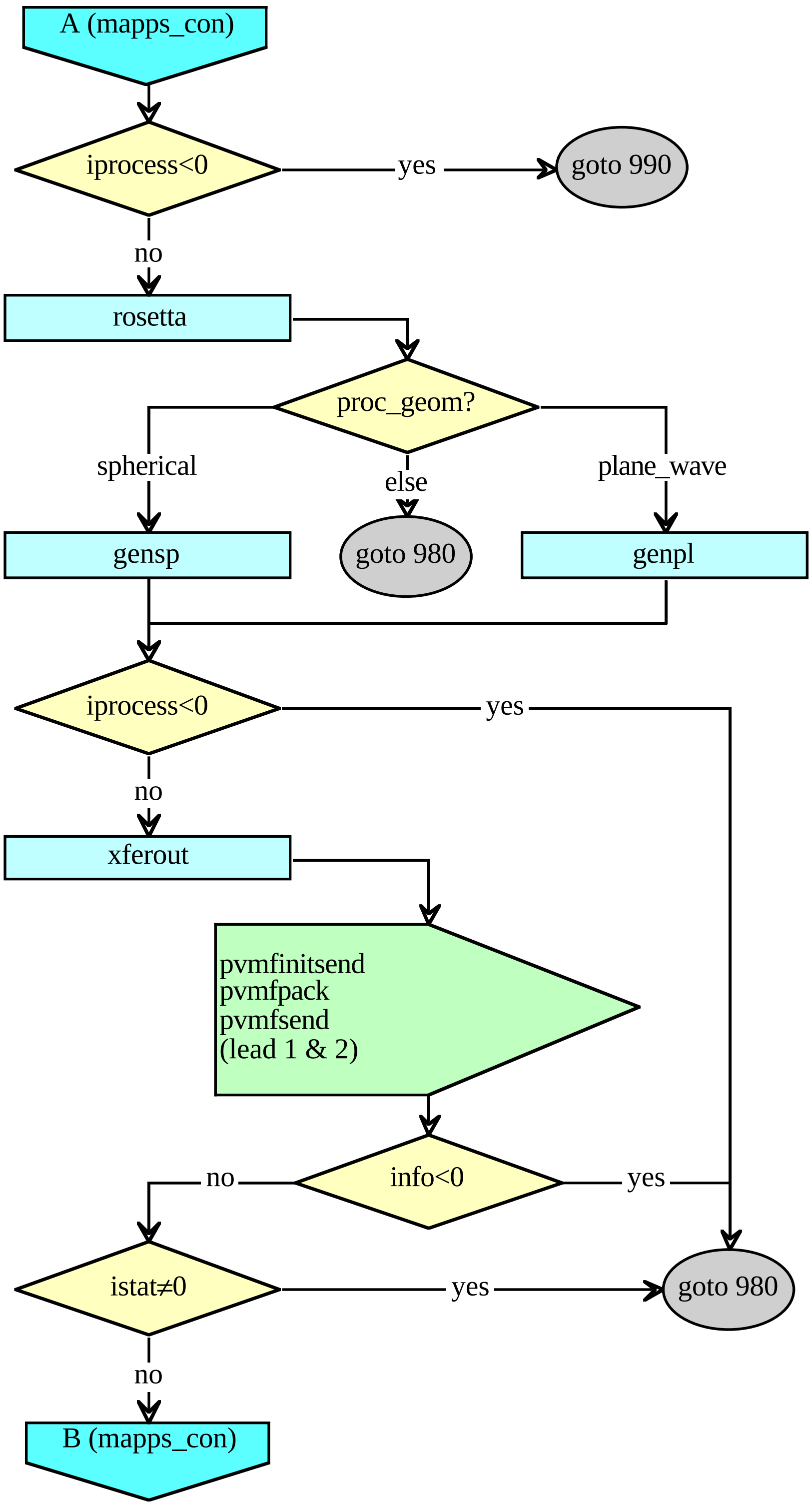
2. Geist, A.; Beguelin, A.; Dongarra, J.; Jiang, W.; Manchek, R.; and Sunderam, V.: PVM 3 User’s Guide and Reference Manual. Engineering Physics and Mathematics Division, Mathematical Sciences Section, prepared by the Oak Ridge National Laboratory, Oak Ridge, TN 37831, ORNL/TM-12187, May 1994.

3. Koga, D. J.; Korsmeyer, D. J.; and Schreiner, J. A.: DARWIN Information System of NASA—An Introduction. AIAA-96-2249, 19th AIAA Advanced Measurement and Ground Testing Technology Conference, New Orleans, LA, June 17–20, 1996.

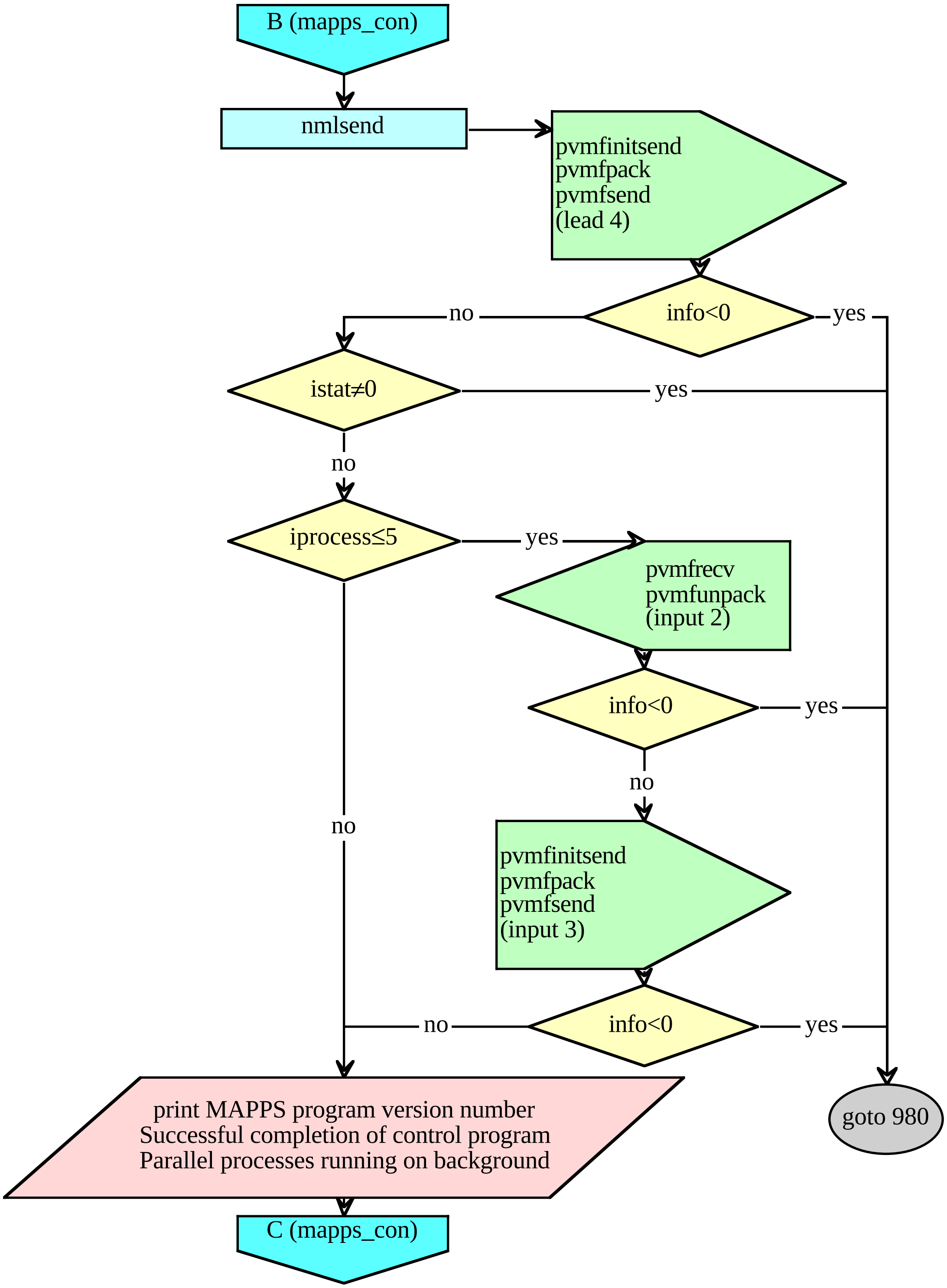
APPENDIX 1. MAPPS PROCESSING SOFTWARE CONTROL PROGRAM FLOW MAP.



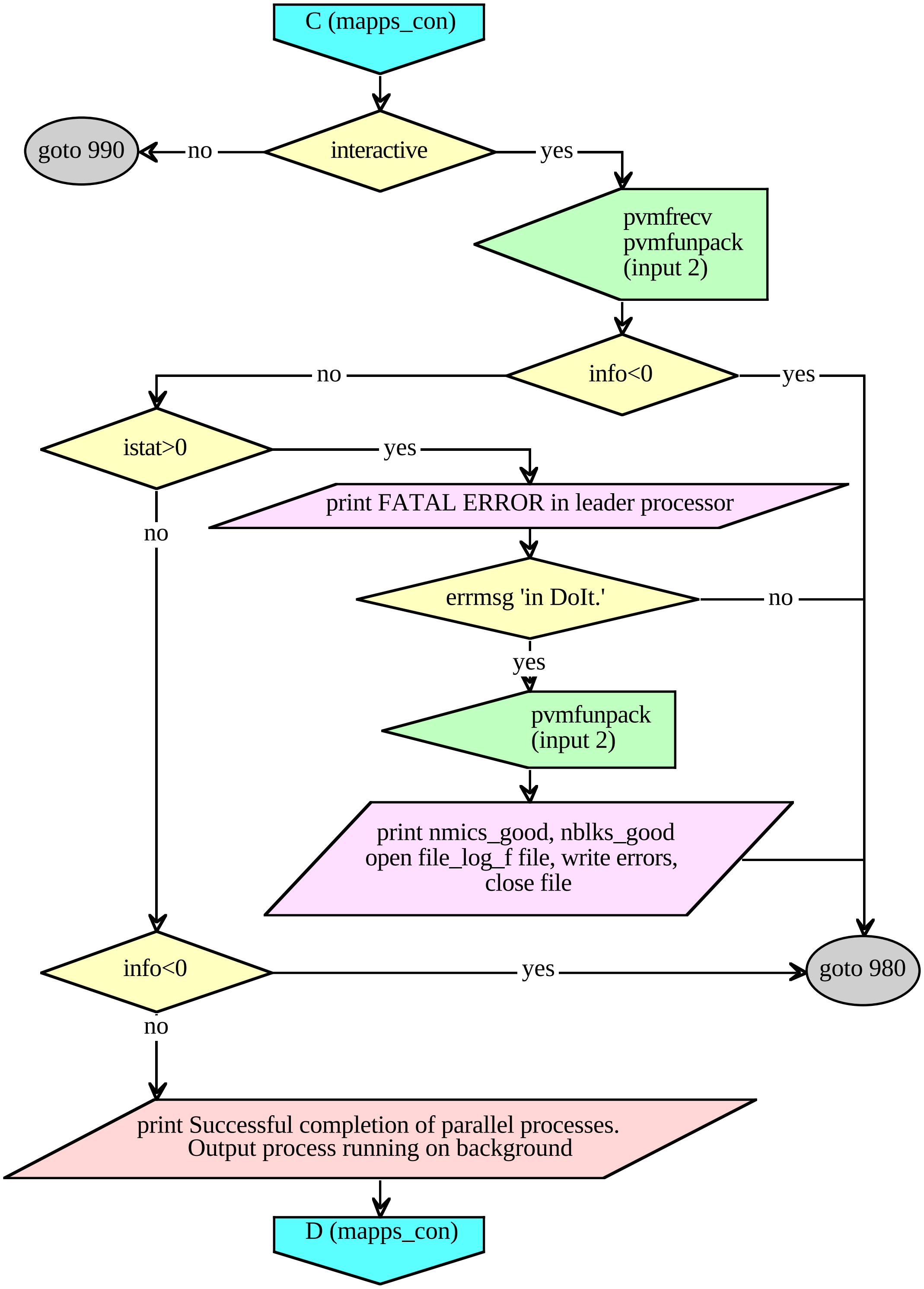
Appendix 1. MAPPS Processing Software Control Program Flow Map (Continued).



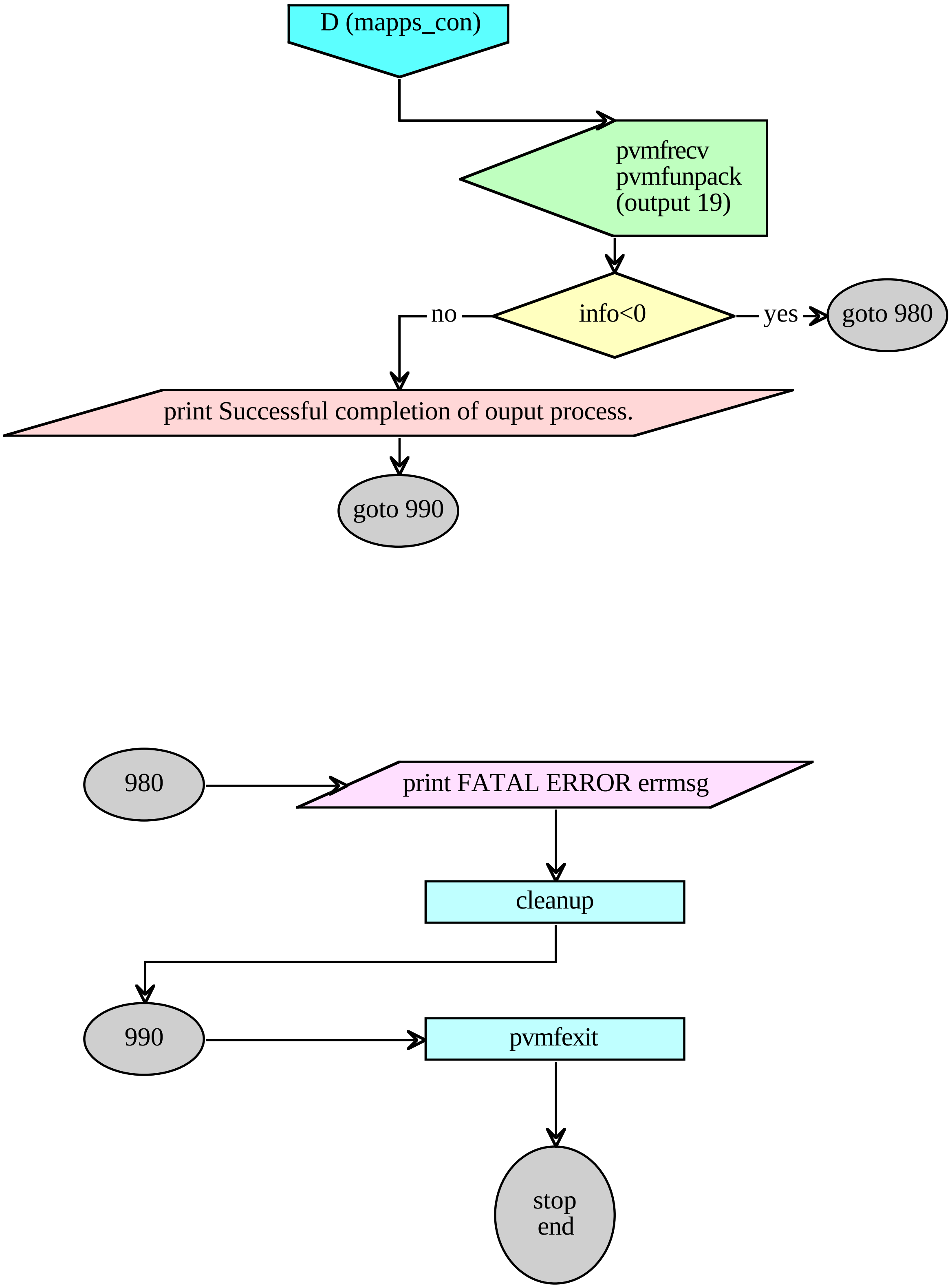
Appendix 1. MAPPS Processing Software Control Program Flow Map (Continued).



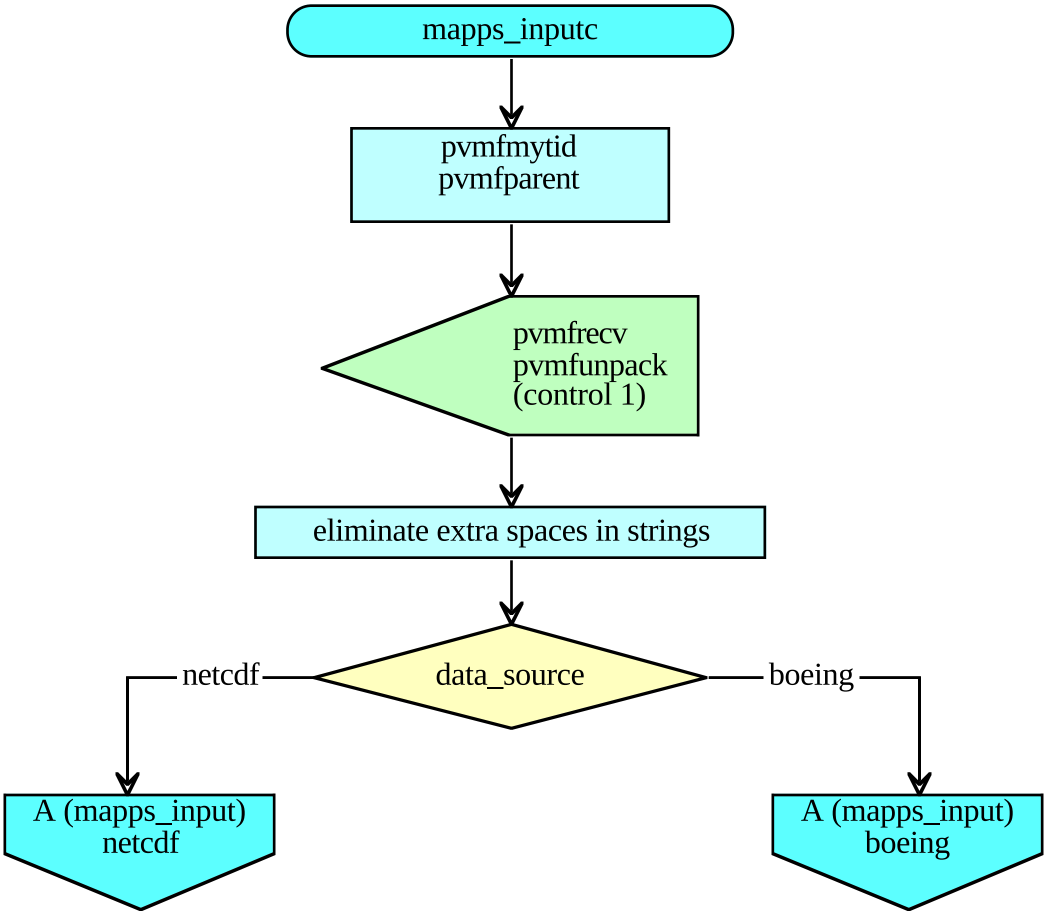
Appendix 1. MAPPS Processing Software Control Program Flow Map (Continued).


C (mapps_con)
interactive
no
goto 990
yes
pvmfrecv
pvmfunpack
(input 2)
info<0
no
yes
istat>0
yes
print FATAL ERROR in leader processor
no
errmsg 'in DoIt.'
no
yes
pvmfunpack
(input 2)
print nmics_good, nblks_good
open file_log_f file, write errors,
close file
info<0
yes
goto 980
no
print Successful completion of parallel processes.
Output process running on background
D (mapps_con)

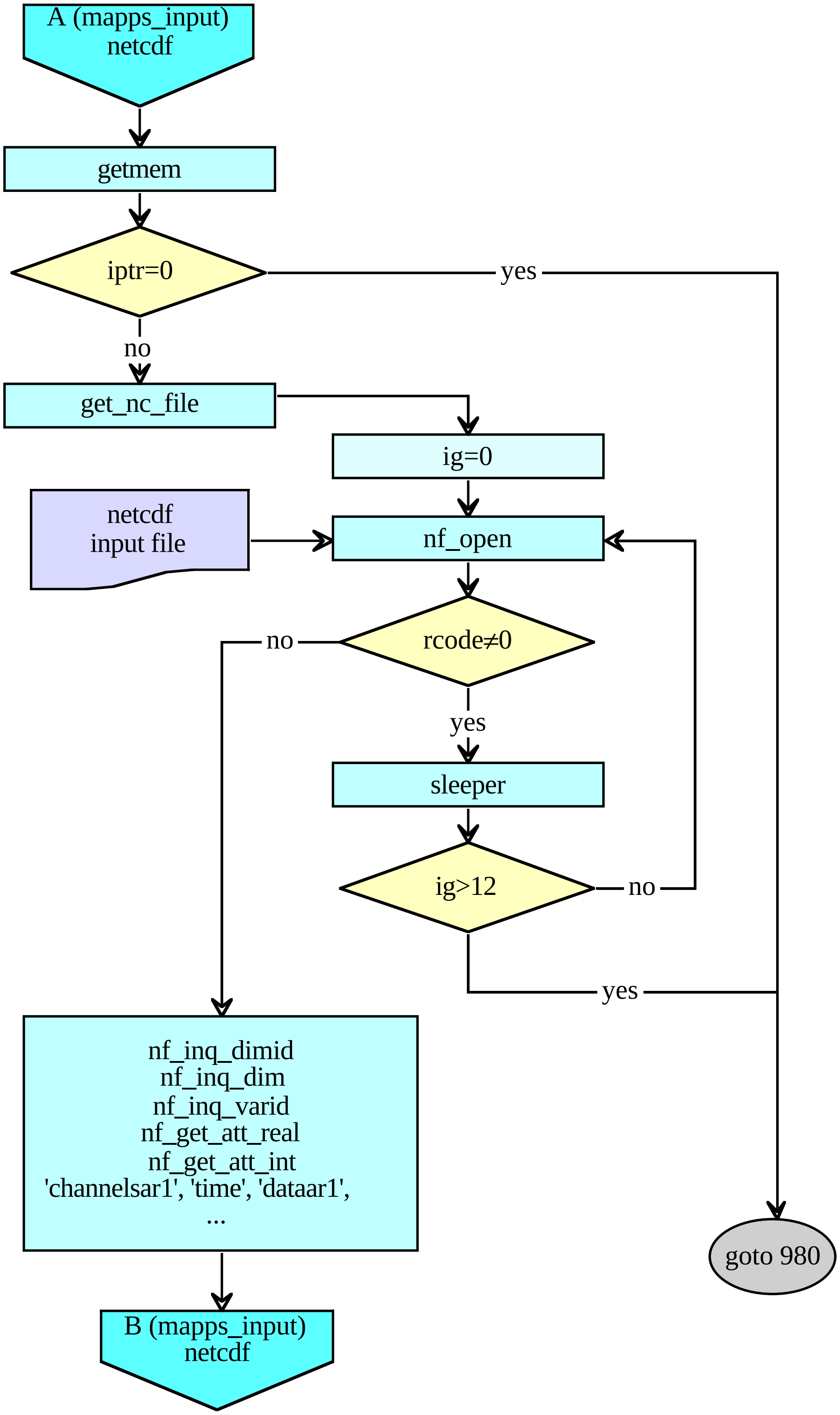
Appendix 1. MAPPS Processing Software Control Program Flow Map (Concluded).



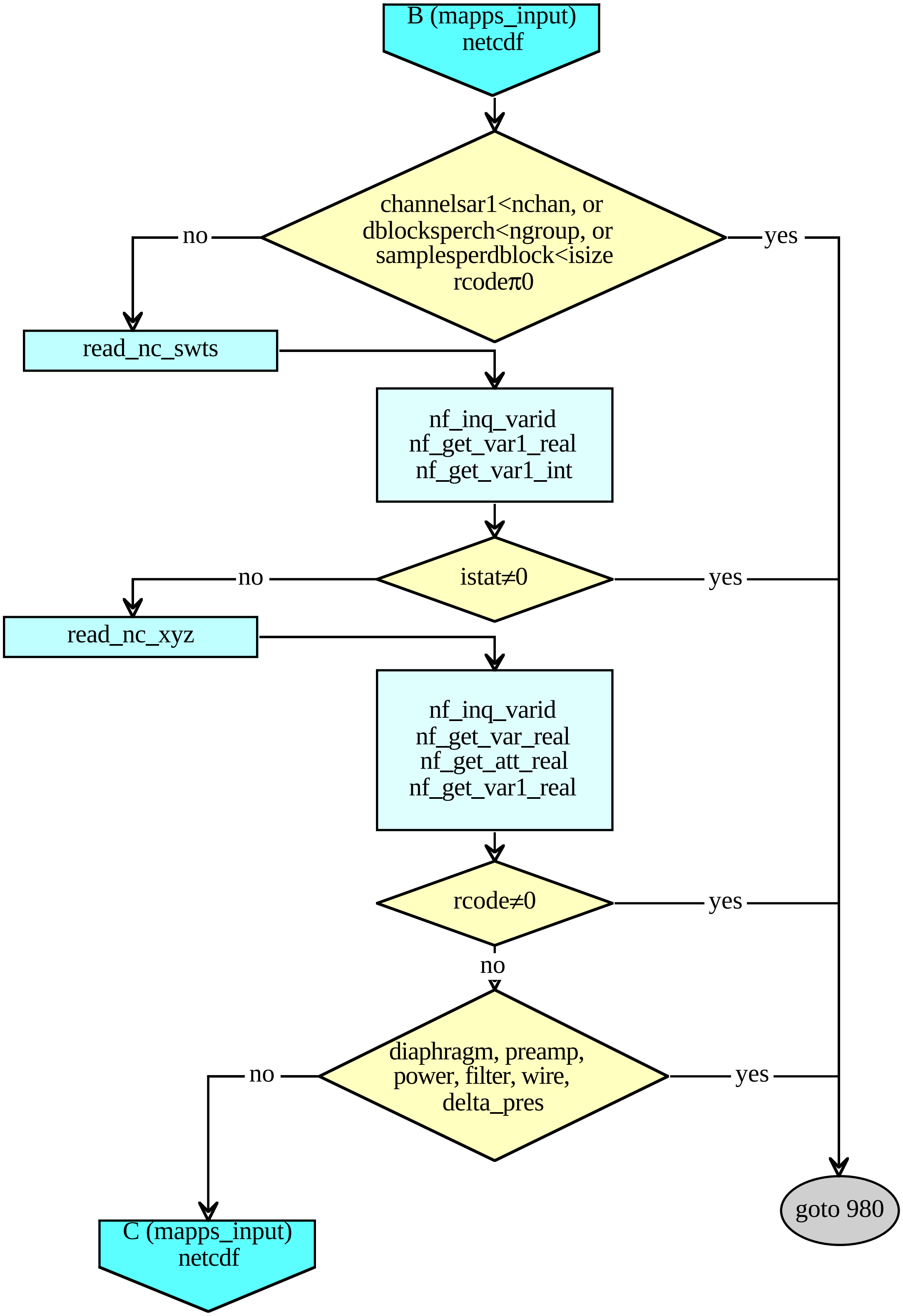
APPENDIX 2. MAPPS INPUT PROGRAM FLOW CHART.



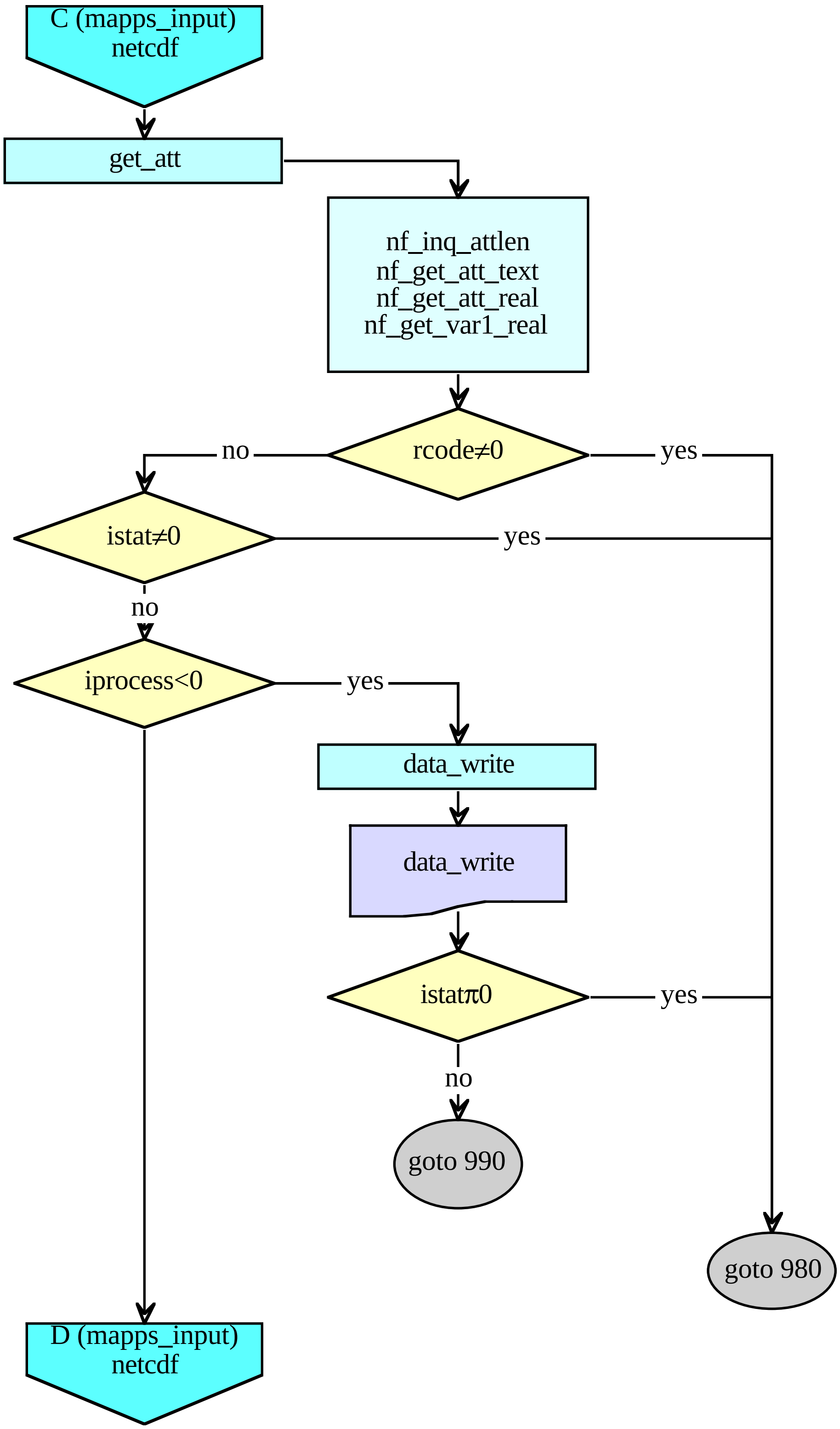
Appendix 2. MAPPS Input Program Flow Chart (Continued).



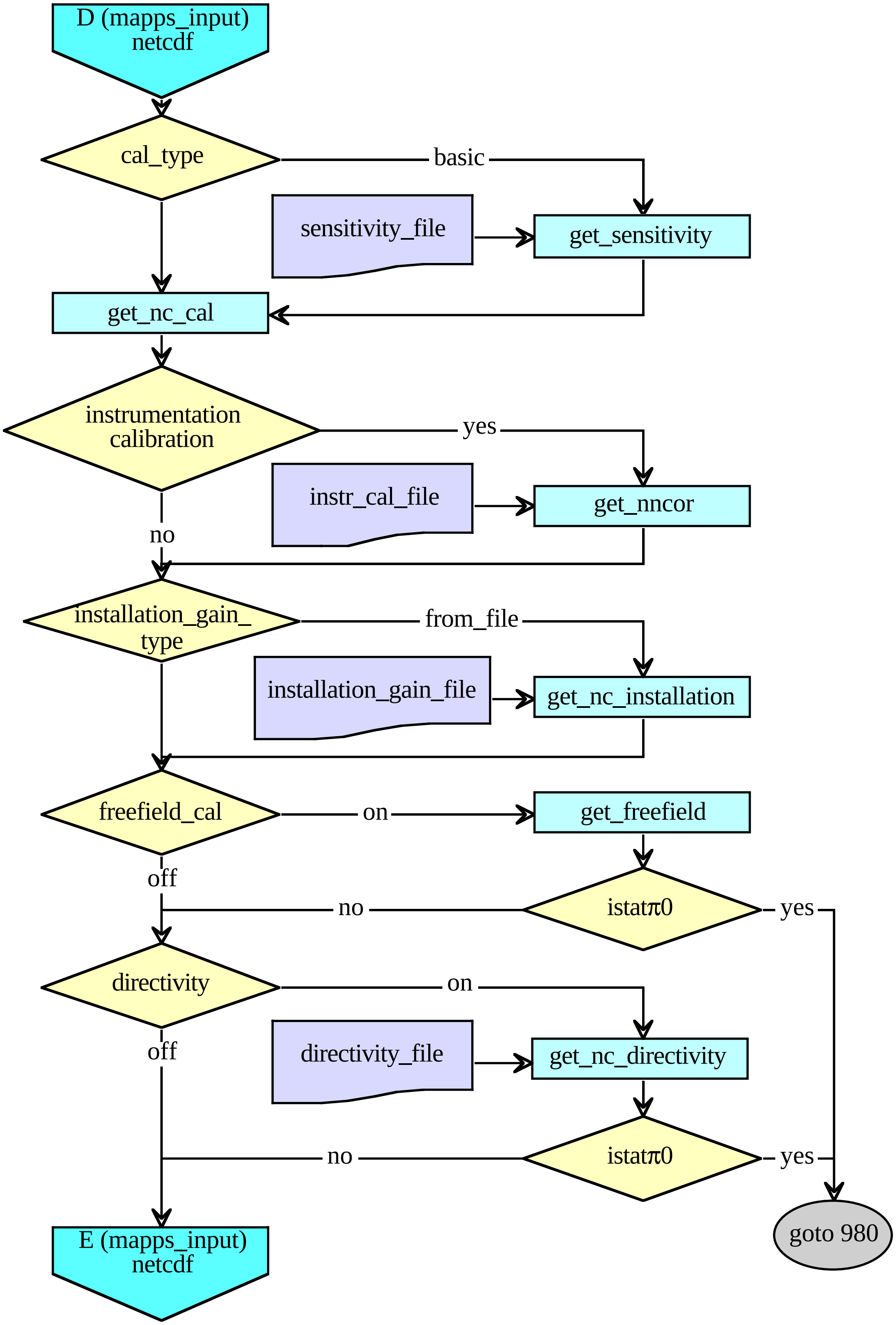
Appendix 2. MAPPS Input Program Flow Chart (Continued).



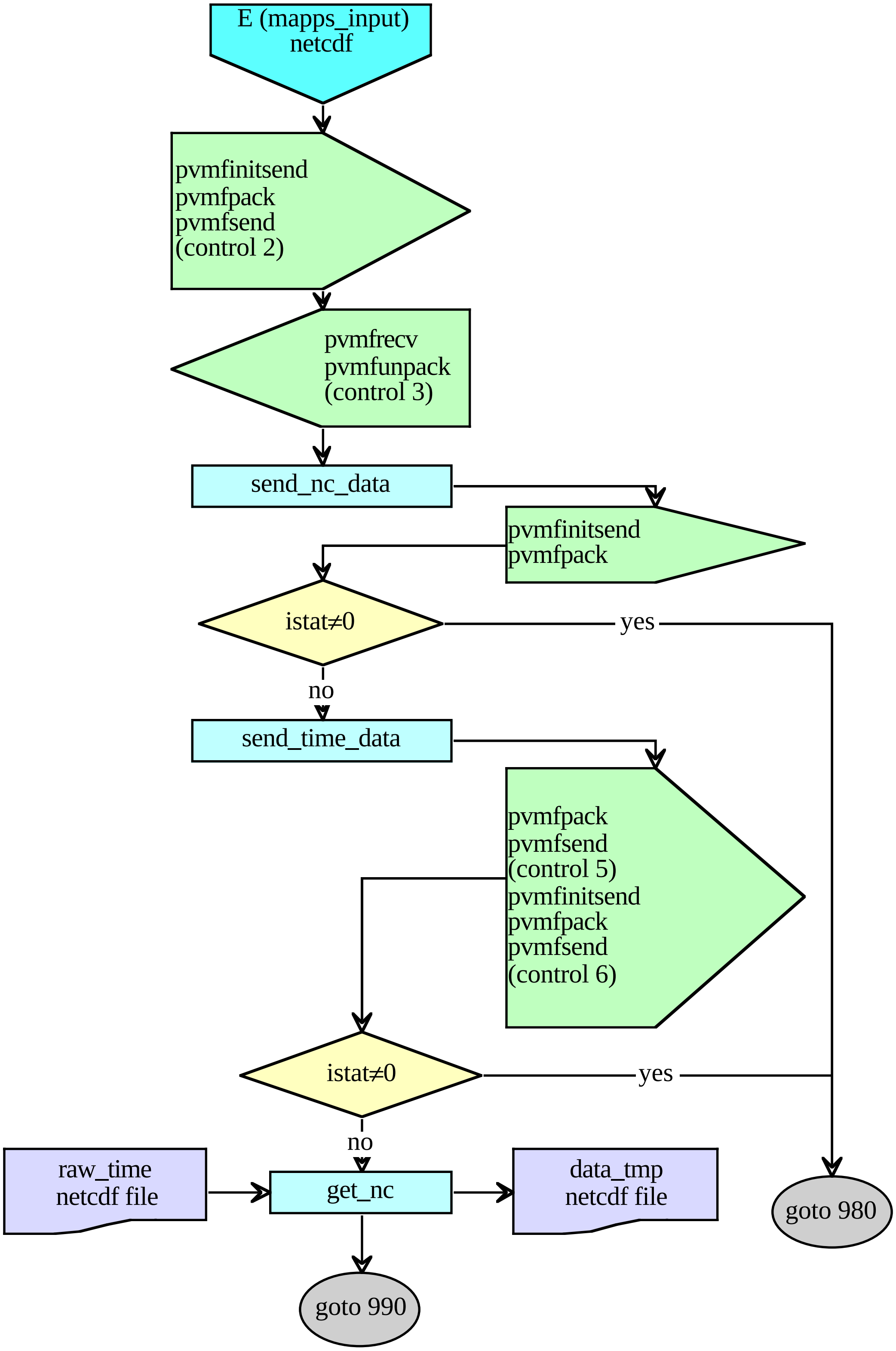
Appendix 2. MAPPS Input Program Flow Chart (Continued).



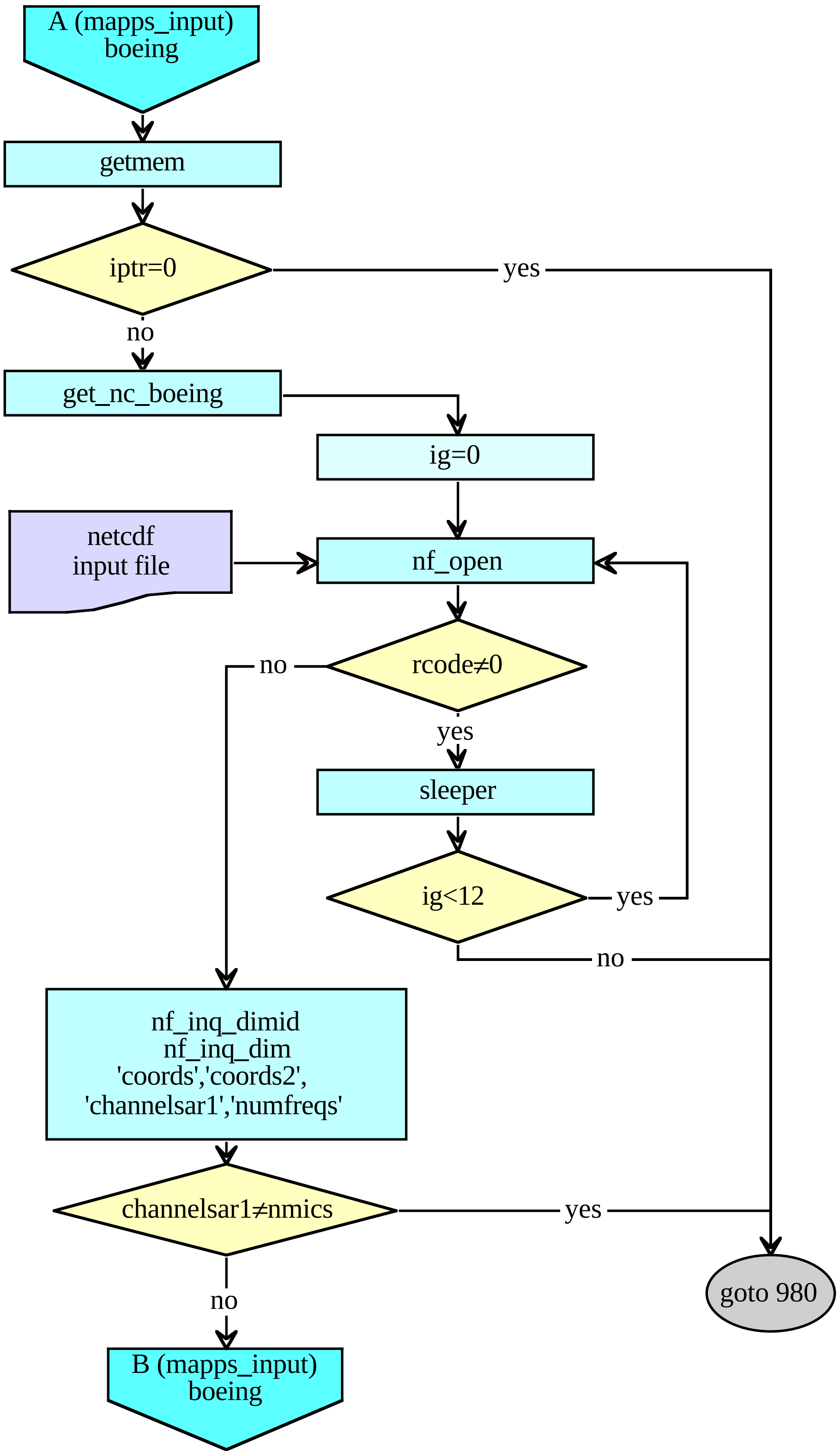
Appendix 2. MAPPS Input Program Flow Chart (Continued).



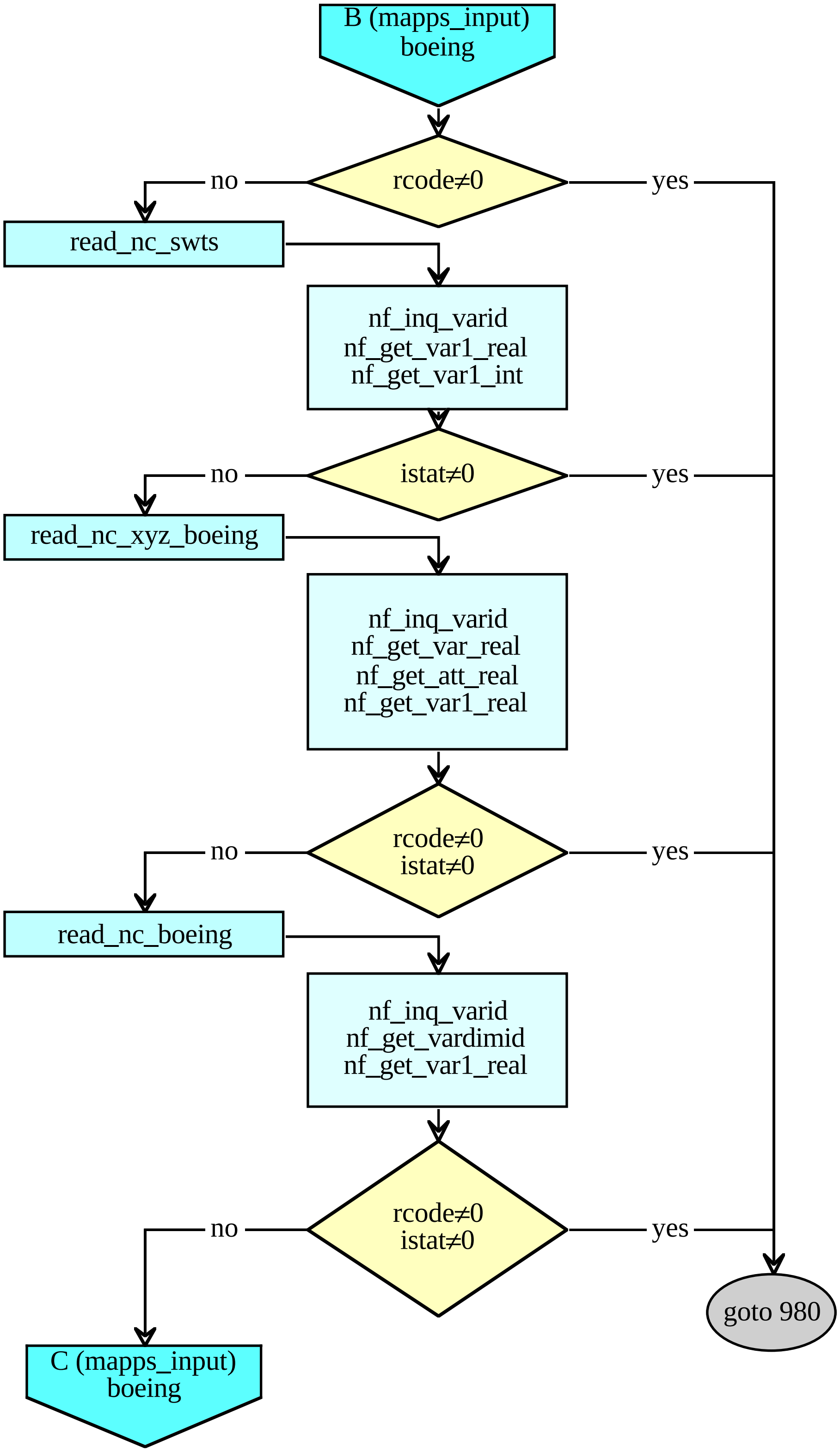
Appendix 2. MAPPS Input Program Flow Chart (Continued).



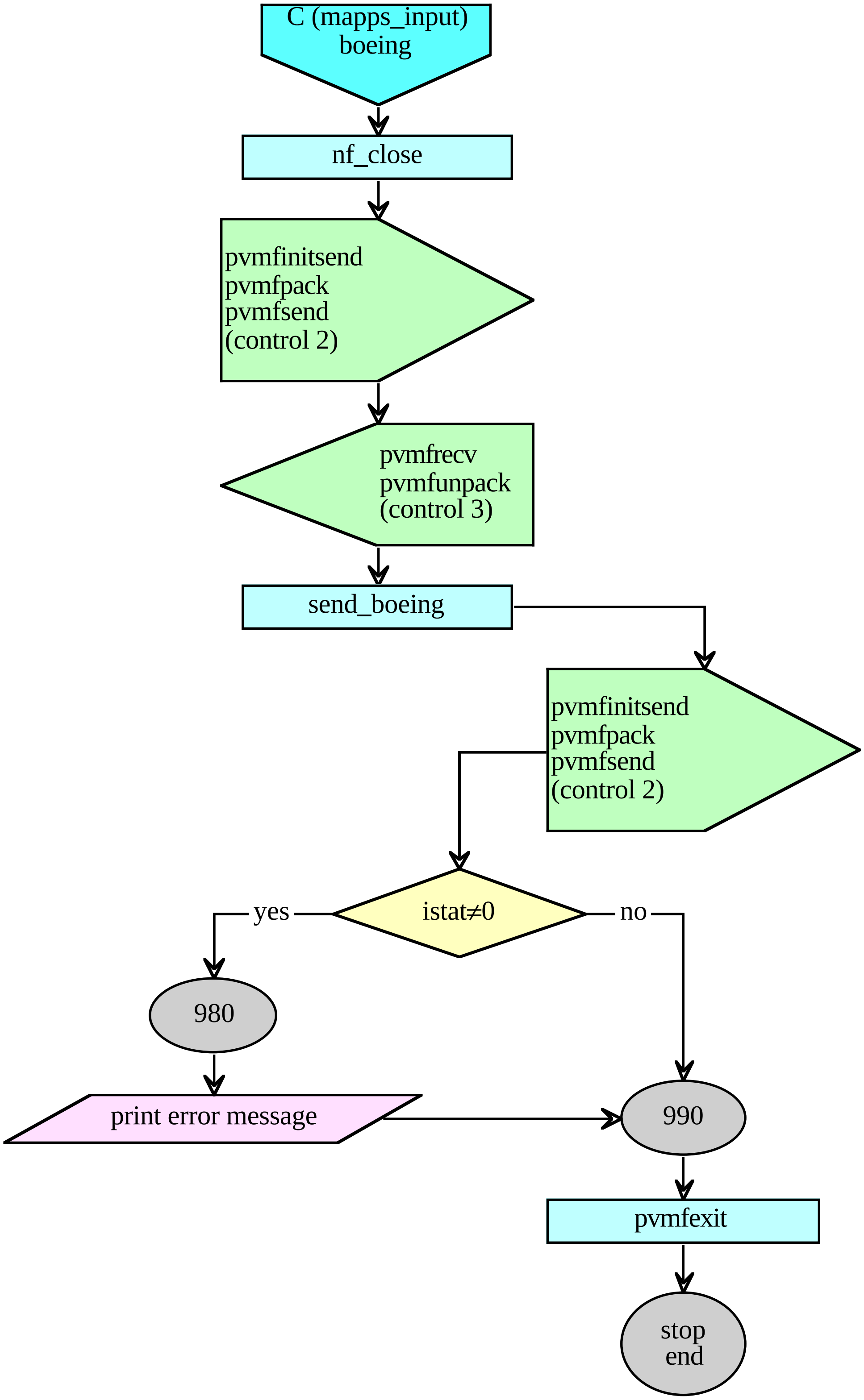
Appendix 2. MAPPS Input Program Flow Chart (Continued).



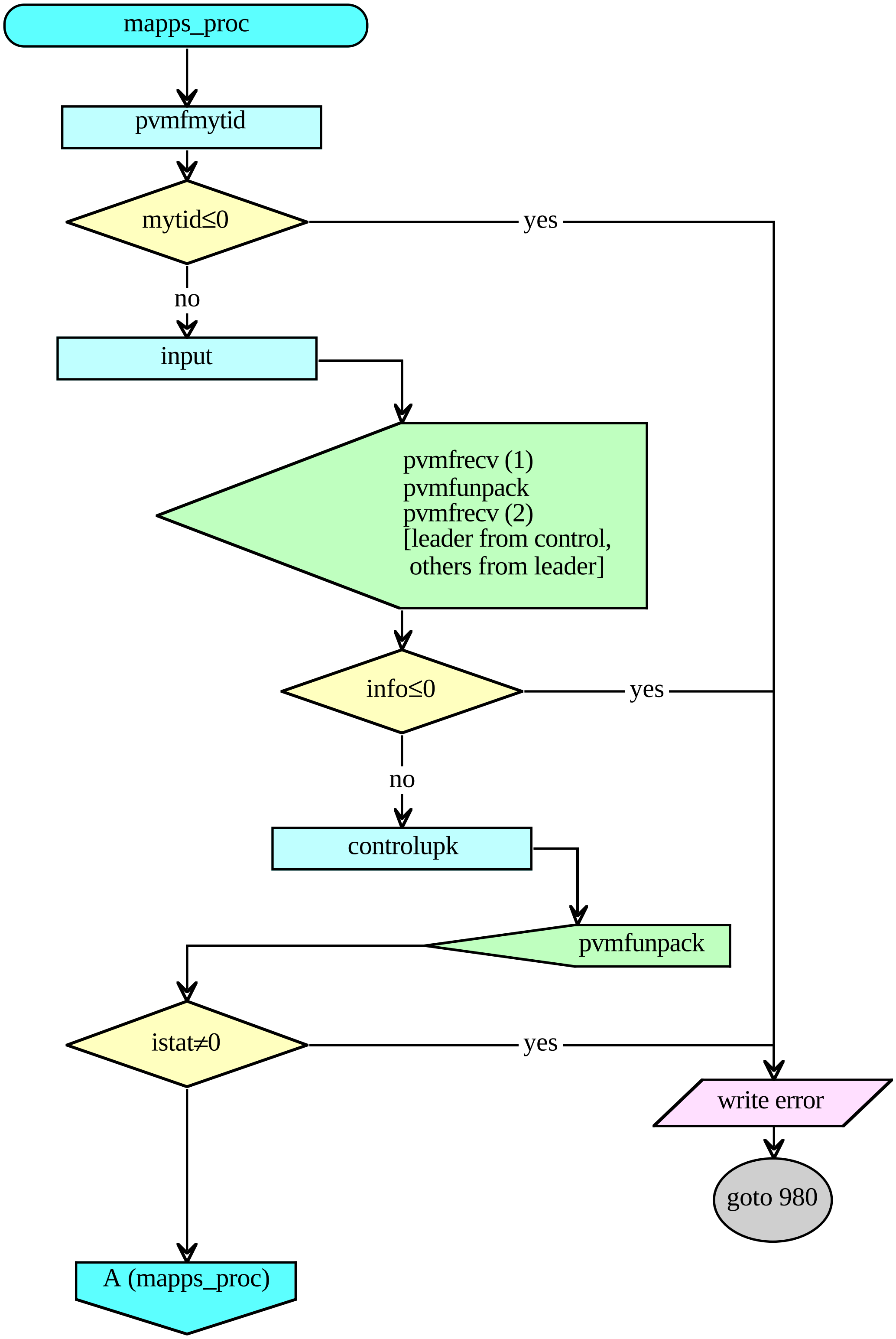
Appendix 2. MAPPS Input Program Flow Chart (Continued).

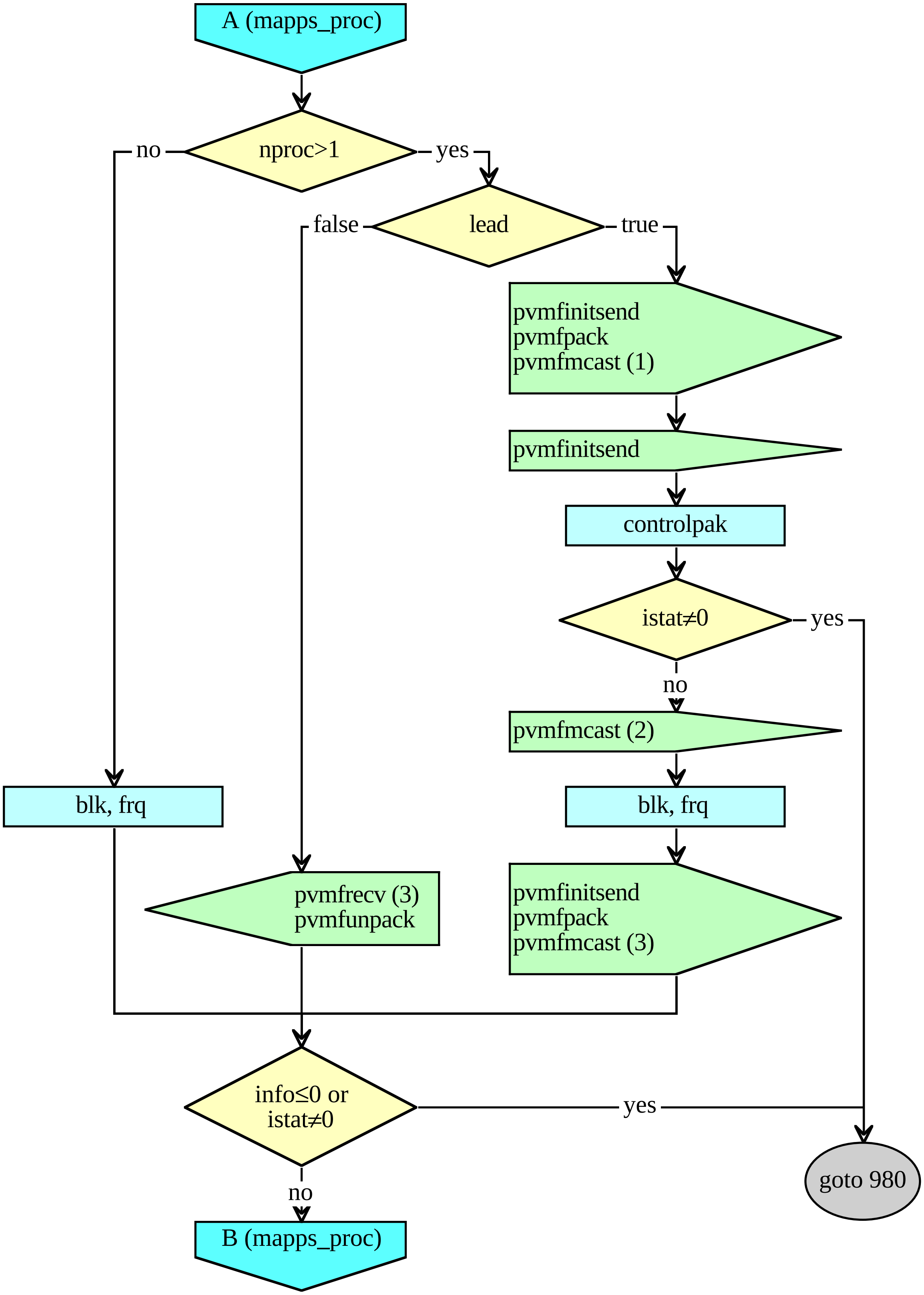


Appendix 2. MAPPS Input Program Flow Chart (Concluded).

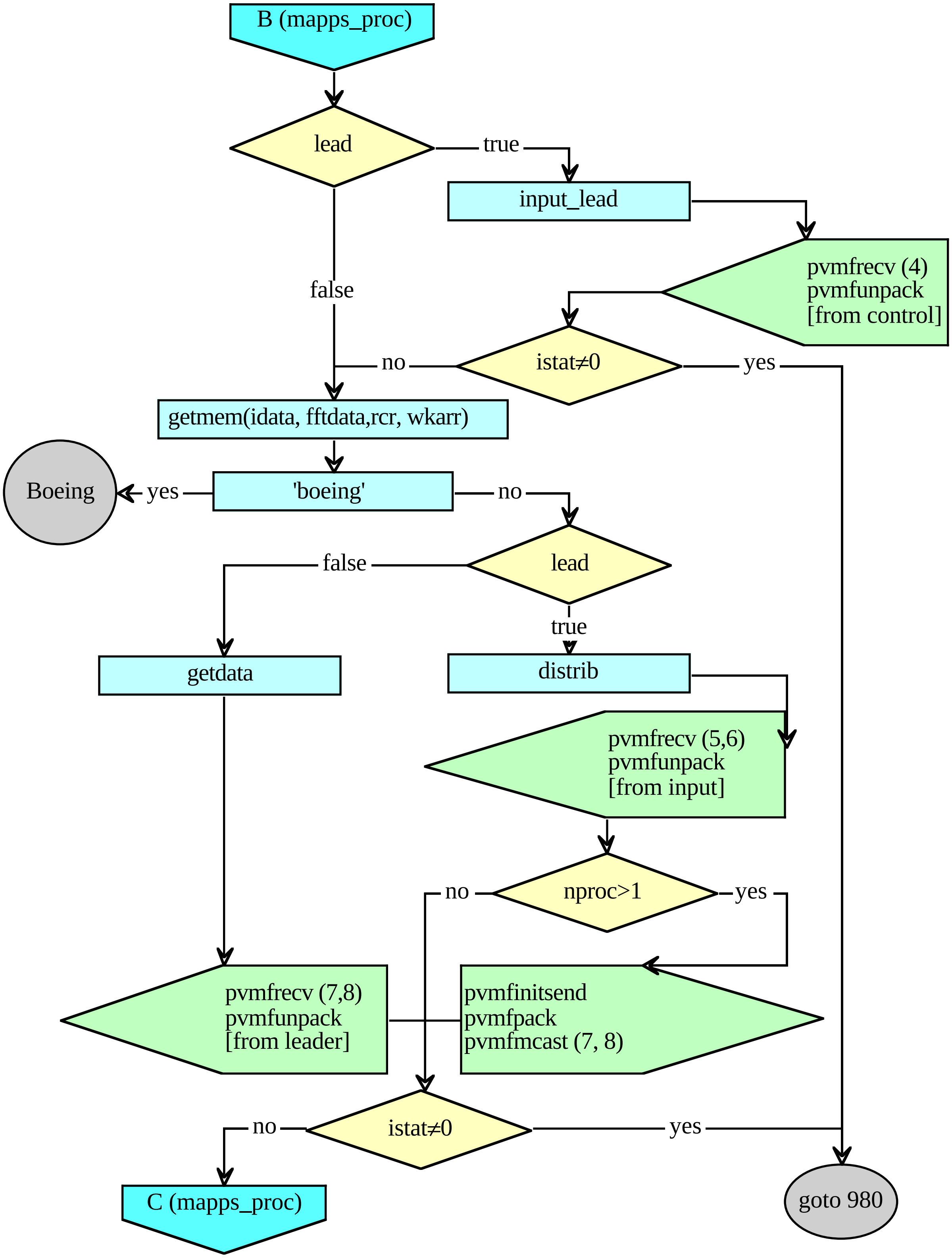


APPENDIX 3. MAPPS PARALLEL PROCESSING PROGRAM FLOW CHART.

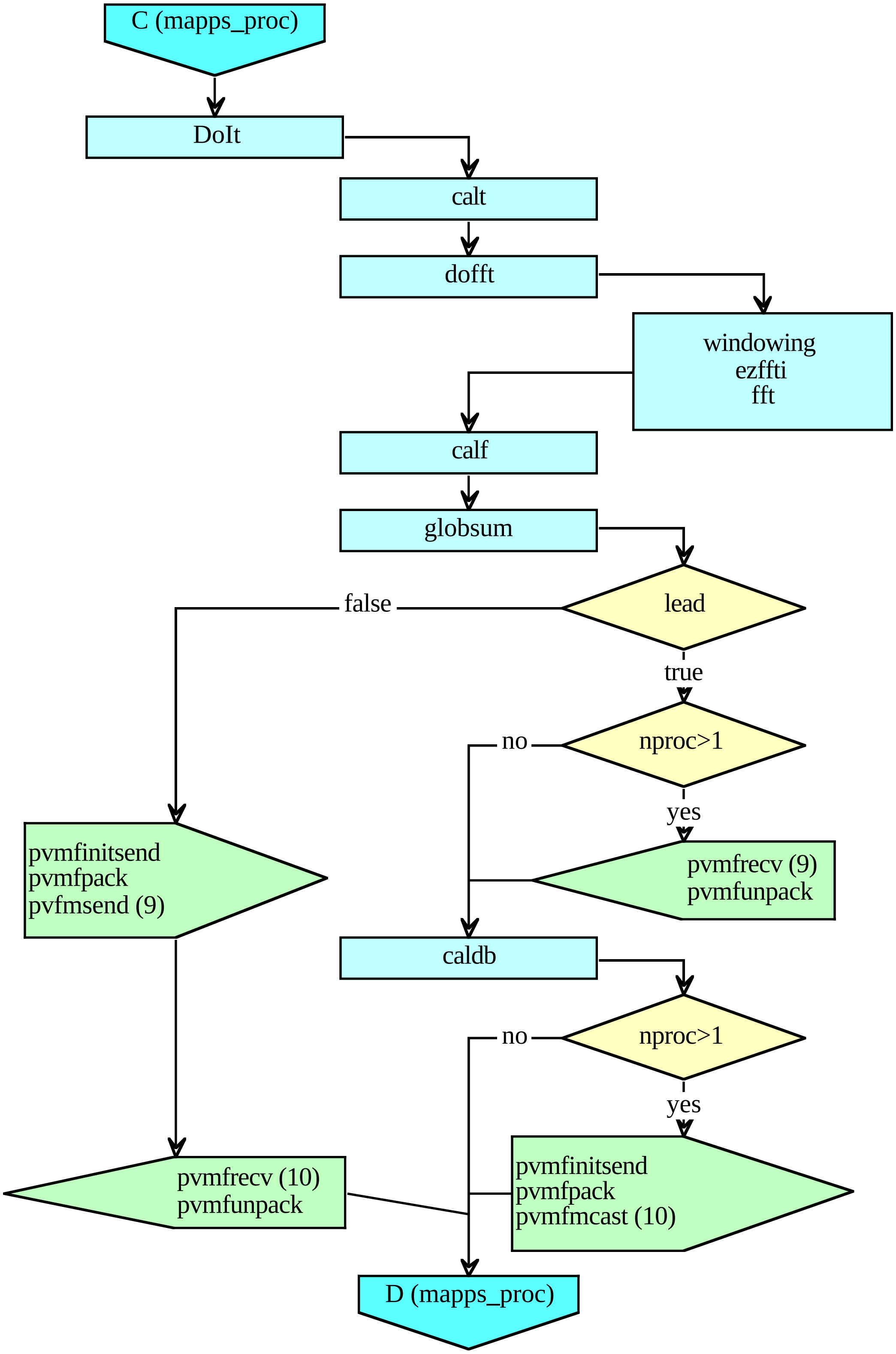



A (mapps_proc)
nproc>1
no
yes
lead
false
true
pvmfinitsend
pvmfpack
pvmfmcast (1)
pvmfinitsend
controlpak
istat≠0
yes
no
pvmfmcast (2)
blk, frq
blk, frq
pvmfrecv (3)
pvmfunpack
pvmfinitsend
pvmfpack
pvmfmcast (3)
info≤0 or
istat≠0
yes
no
goto 980
B (mapps_proc)

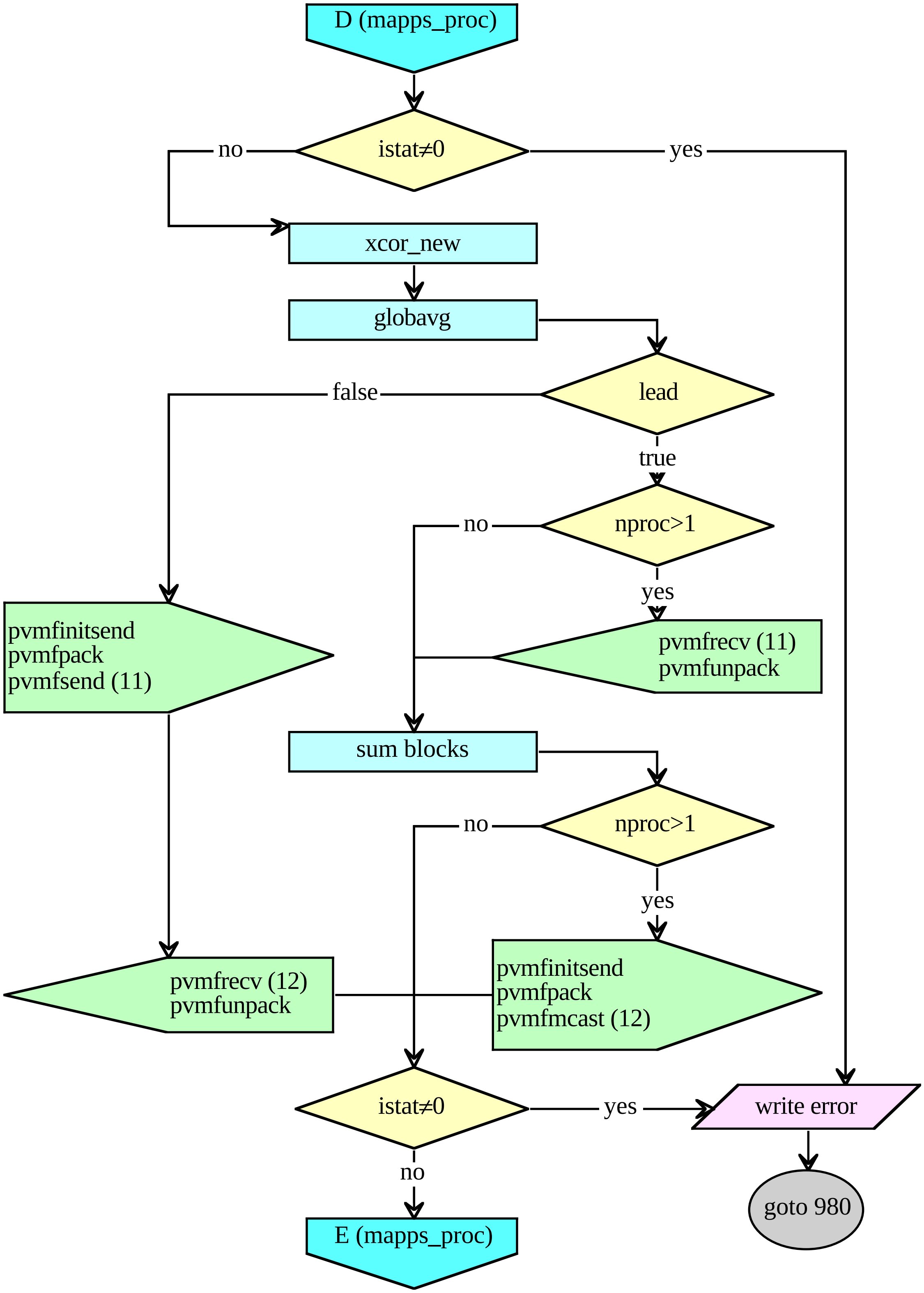
Appendix 3. MAPPS Parallel Processing Program Flow Chart (Continued).



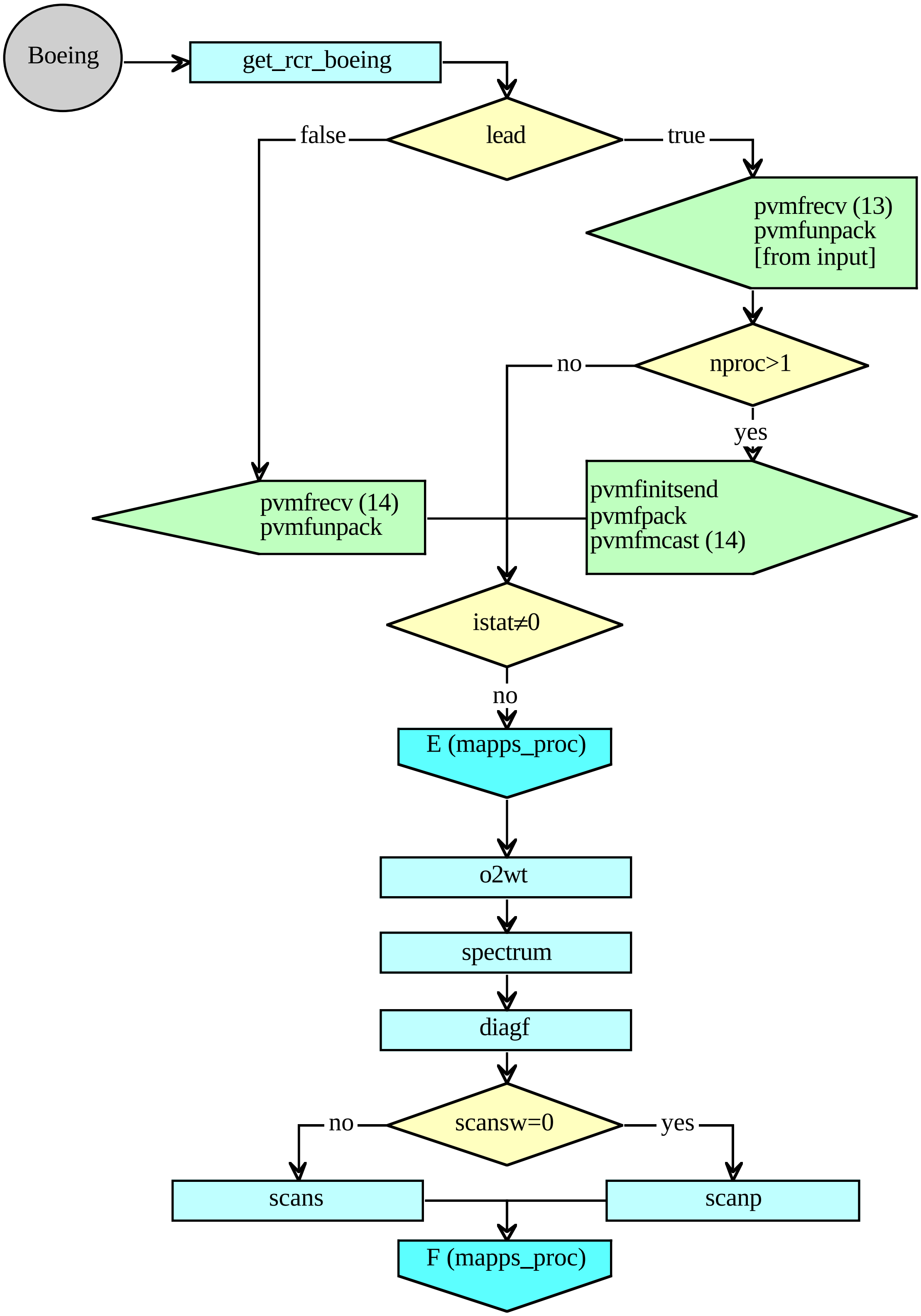
Appendix 3. MAPPS Parallel Processing Program Flow Chart (Continued).



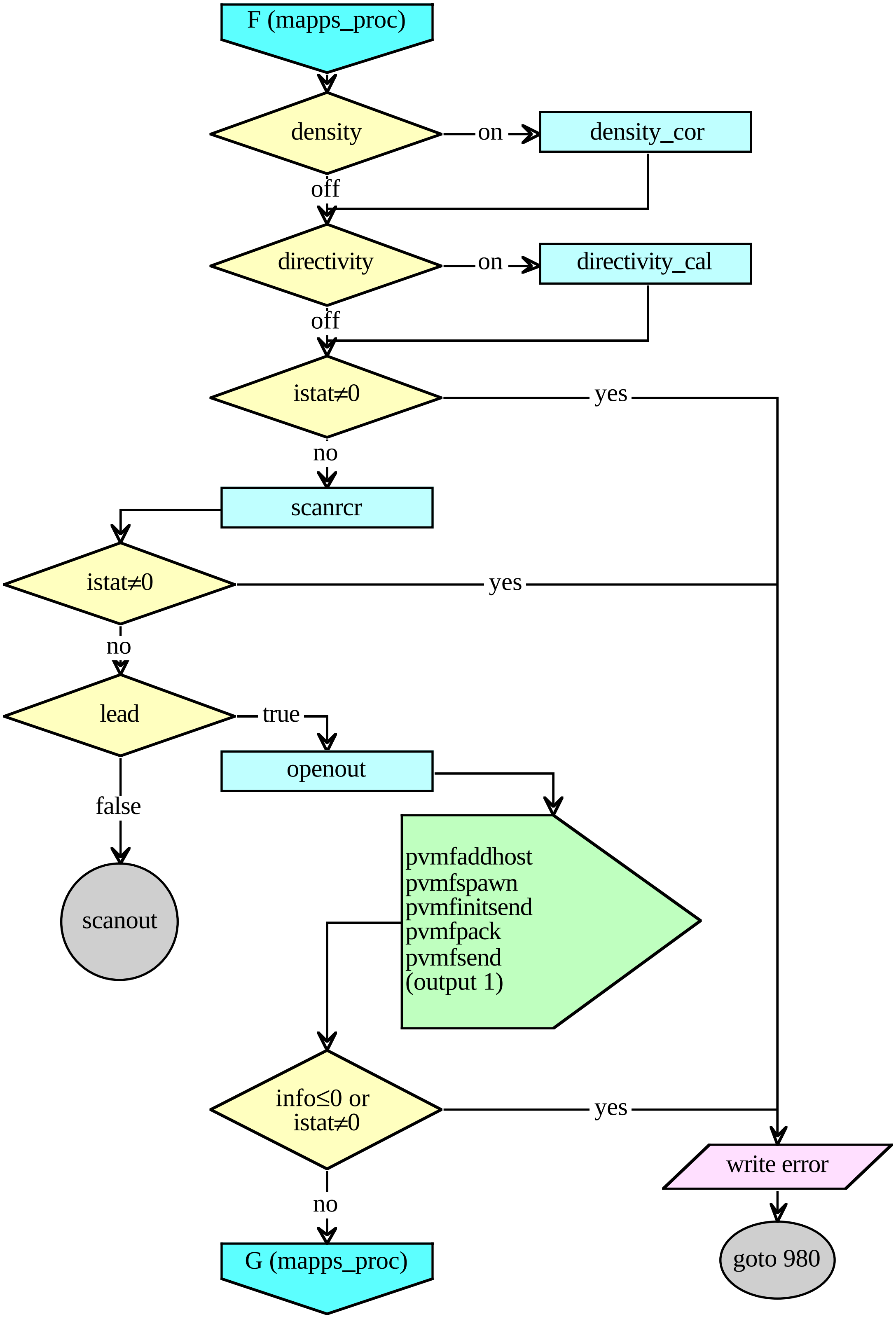
Appendix 3. MAPPS Parallel Processing Program Flow Chart (Continued).



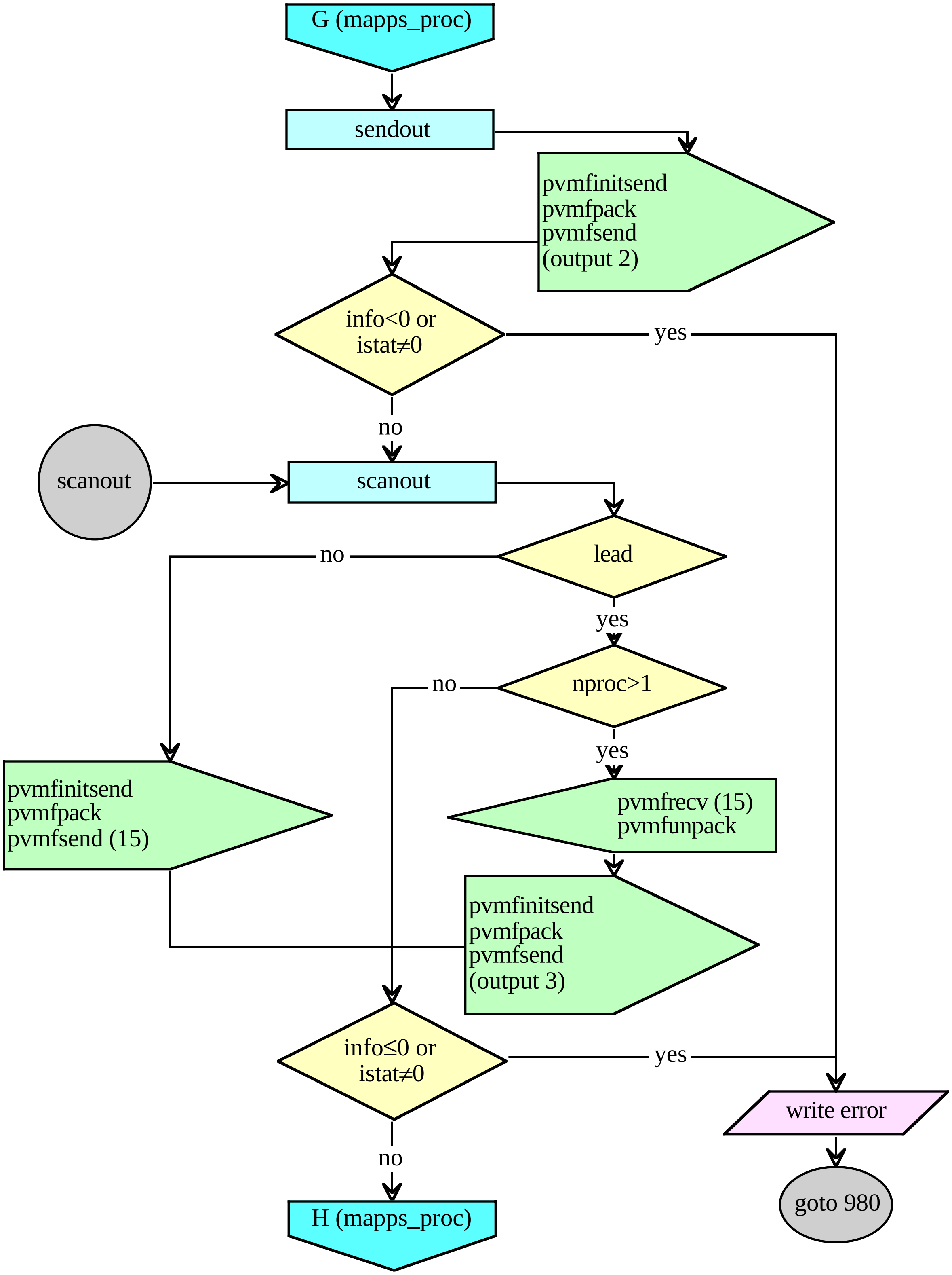
Appendix 3. MAPPS Parallel Processing Program Flow Chart (Continued).



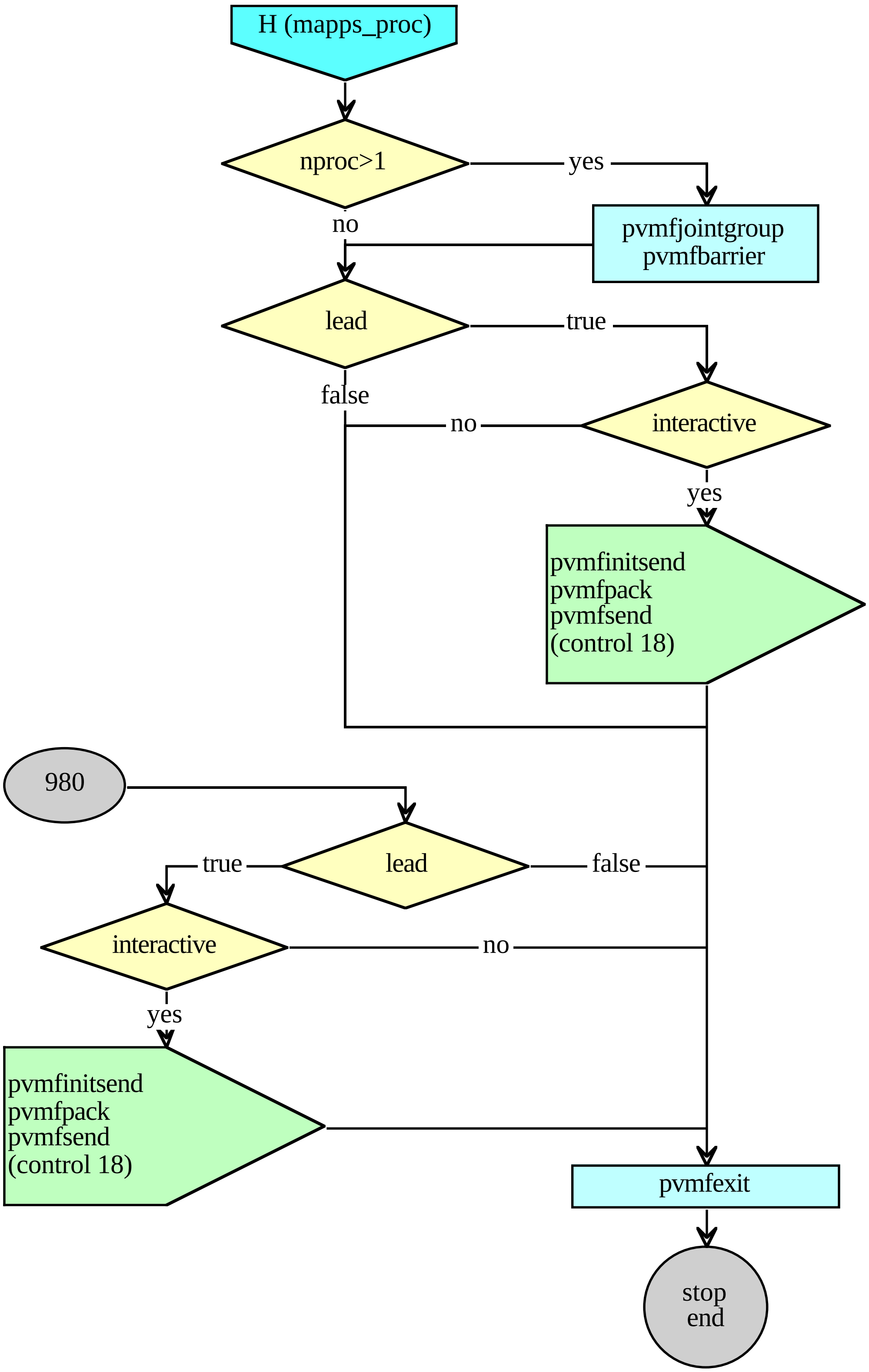
Appendix 3. MAPPS Parallel Processing Program Flow Chart (Continued).



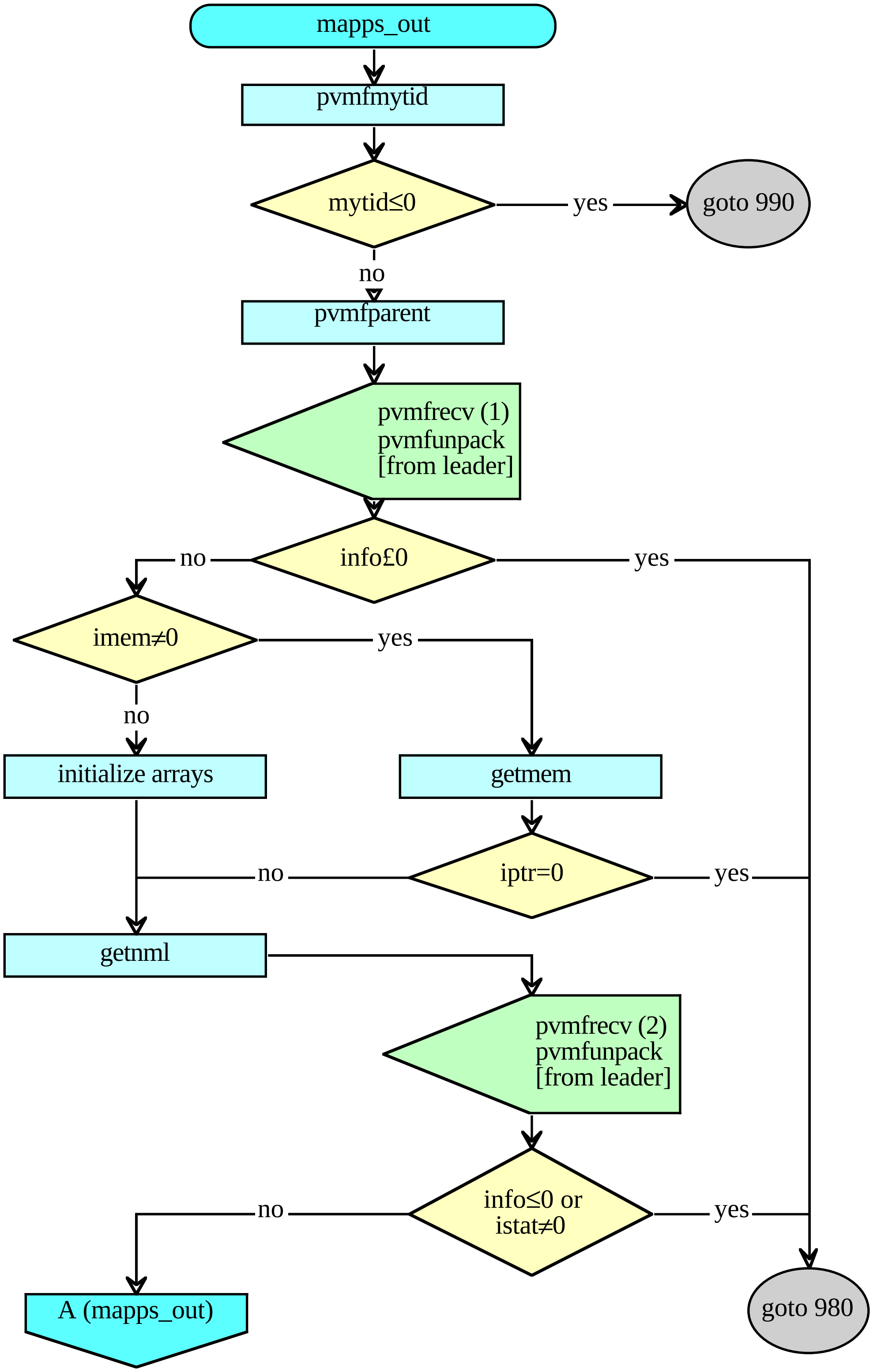
Appendix 3. MAPPS Parallel Processing Program Flow Chart (Continued).



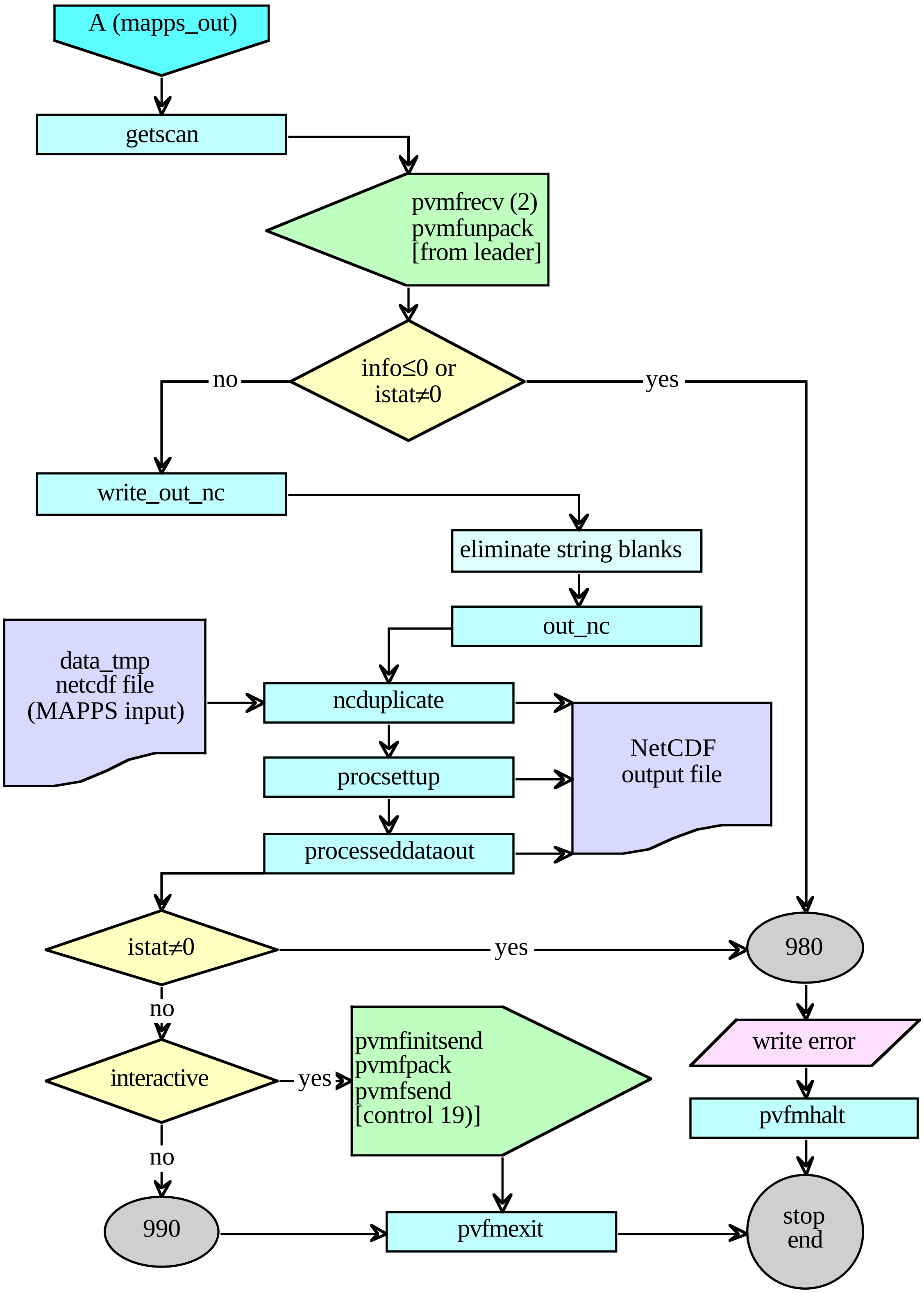
Appendix 3. MAPPS Parallel Processing Program Flow Chart (Concluded).



APPENDIX 4. MAPPS OUTPUT PROGRAM FLOW CHART.



Appendix 4. MAPPS Output Program Flow Chart (Concluded).



APPENDIX 5. RAW DATA FORMAT DESCRIPTION

The netCDF raw data file content is described in this appendix. An example of a raw time history file is shown in appendix 2. Variables are indicated in bold type with their attributes in normal type. The type and dimensions of the variables and their attributes are specified as [type] [size] following the description.

dimensions:

coords = 3	number of coordinates for vectors [int]
coords2 = 9	number of coordinate transformation matrix (3x3) mapped into vector format [int]
channelsar1	number of VXI channels [int]
numvxicards	number of VXI A/D cards [int]
time	number of time samples for each channel [int]

variables:

test	test number [int]
run	run number [int]
point	point number for the servio system [int]
array_point	point number specific to the array [int]
wt_data_quality	flag for status of wind tunnel condition data contained in this file, 0 = good, 1 = approximate, 2 = absent [int]
facility	facility key: 0=other, 1=acoustic lab, 2=12-ft, 3=40x80, 4=80x120, 5=OARF, 6=unitary, 7=7x10 [int]
mtunxdir	tunnel Mach number x axis direction: +1=positive pointing downstream, –1=positive pointing upstream [int]
maingr	model main gear indicator, 1=ON, 0=OFF [int]
nosegr	model nose gear indicator, 1=ON, 0=OFF [int]
baro	tunnel barometric pressure in psi [float]
rh	tunnel relative humidity in % [float]
qc	tunnel corrected dynamic pressure in psf [float]
ptopsf	tunnel total pressure in psf [float]
psopsf	tunnel static pressure in psf [float]
ttr	tunnel total temperature in °R [float]
tsr	tunnel static temperature in °R [float]
vfps	tunnel velocity in ft/s [float]
vkts	tunnel velocity in knots [float]
cfps	tunnel speed of sound in ft/s [float]
mtun	tunnel Mach number [float]
inc1	model angle of incidence measured at X in deg. [float]
bflap	model inboard flap setting in deg. [float]
eslat	model slat setting in deg. [float]
modelscale	model scale factor [float]

modelrefxyzwtc model coordinate system origin reference x,y,z location in inches in wind tunnel coordinate system [float] [coords]

modelrefrotwtc model reference rotation matrix from model to wind tunnel coordinate system stored as a vector (first 3 elements of vector are top row of rotation matrix) [float] [coords2]

modelrefangnat model natural angles (pitch, roll, yaw) in model natural coordinate system in deg. [float] [coords]

modeldeltaxyzwtc model coordinate system delta x, y, z from reference in wind tunnel coordinate system in inches [float] [coords]

modelabsxyzwtc model coordinate system origin absolute x, y, z location in wind tunnel coordinate in inches [float] [coords]

modelabsrotwtc model absolute rotation matrix from model to wind tunnel coordinate system stored as a vector (first 3 elements of vector are top row of rotation matrix) [float] [coords2]

miclocsar1 microphone coordinates for array #1 in inches in array coordinate system [float] [channelsar1, coords]

- long_name long name for the microphone array [char*]
- comment user comment [char*]
- units coordinate system units [char*]
- arrayPatternID microphone array pattern ID [char*]
- locID location ID of the channels in array pattern [int] [channelsar1]
- arrayPatternVersion array pattern version comment [char*]
- arrayrefxyzwtccom arrayrefxyzwtc descriptive comment [char*]
- arrayrefxyzwtc array coordinate system reference x, y, z position in inches in wind tunnel coordinate system [float] [coords]
- arrayrefrotwtccom arrayrefrotwtc descriptive comment [char*]
- arrayrefrotwtcformat arrayrefrotwtc format description [char*]
- arrayrefrotwtc array coordinate system reference rotation matrix from array to wind tunnel coordinate system stored as a vector (first 3 elements of vector are top row of rotation matrix) [float] [coords2]
- arrayrefangnatcom arrayrefangnat descriptive comment [char*]
- arrayrefangnat array reference angles (xi, zeta, upsilon) in array natural coordinate system in degrees. [float] [coords]
- arraydeltaxyzwtccom arraydeltaxyzwtc descriptive comment [char*]
- arraydeltaxyzwtc array coordinate system x, y, z delta from reference position in inches in wind tunnel coordinates [float] [coords]
- arrayabsxyzwtccom arrayabsxyzwtc descriptive comment [char*]
- arrayabsxyzwtc array coordinate system absolute x, y, z position in wind tunnel coordinate in inches [float][coords]
- arrayabsrotwtccom arrayabsrotwtc descriptive comment [char*]
- arrayabsrotwtcformat arrayabsrotwtc format description [char*]
- arrayabsrotwtc array coordinate system absolute rotation matrix from array to wind tunnel coordinate system stored as a vector (first 3 elements of vector are top row of rotation matrix) [float][coords2]
- forbodyID microphone fore-body ID [int] [channelsar1]
- micmancom micman comment

micman microphone manufacturer indicator where 0=other, 1=GRASS, 2=B&K [int] [channelsar1]

micmodel microphone model string composed of model name for each channel separated by a | character [char*]

micserial microphone serial number string composed of a serial number for each channel separated by a | character [char*]

preampmancom preampman comment [char*]

preampman preamp manufacturer indicator where 0=other, 1=GRASS, 2=B&K [int] [channelsar1]

preampmodel preamplifier model string composed of preamplifier model name for each channel separated by a | character [char*]

preampserial preamplifier serial number string composed of a serial number for each channel separated by a | character [char*]

powermodel power supply model string composed of powersupply model name for each channel separated by a | character [char*]

powerchasslotchan power supply box, slot, channel location indicator number composed of box number*10000 + slot number*100 + channel number [int] [channelsar1]

extfiltmancom extfiltman comment [char*]

extfiltman external filter manufacturer indicator where 0 = Pacific Instruments [int] [channelsar1]

extfilmodel external filter model string composed of filter model name for each channel separated by a | character [char*]

extfiltserial external filter serial number string composed of a serial number for each channel separated by a | character [char*]

micabsar1 microphone absolute positions (x, y, z) in inches in wind tunnel coordinate system for array #1 [float] [channelsar1, coords]

long_name long name for micabsar1 [char*]

units units [char*]

dataar1 array #1 time history data [short] [time, channelsar1]

long_name long name for array 1 data[char*]

comment dataar1 comment [char*]

dblocksperch data blocks per channel [int]

samplesperdblock number of time samples per data block per channel [int]

skiptime skipped time between data blocks in seconds [float]

freqsamp effective sampling frequency in samples per second [float]

freqspan user specified upper frequency of interest in Hz [float]

intfilthigh internal vxi high-pass filter value in Hz [float]

intfiltlow internal vxi low-pass filter value in Hz [float]

trigmode triggering mode where A = auto, M = manual [char]

trigdelay trigger delay in seconds [float]

trigext external trigger on, 1=TRUE, 0=FALSE [int]

triglevelup external trigger upper level in volts [float]

triglevellow external trigger lower level in volts [float]

fftwindow vxi FFT widowing where N=no window (rectangular), H=Hanning [char]

units	dataar1 units [char*]
engunits	engineering units [char*]
xducertypecom	transducer type descriptive comment [char*]
xducertype	transducer type indicator where 0=other, 1=array microphone, 2=pressure transducer, 3=reference microphone, 4=reference signal, 5=single microphone [int] [channelsar1]
xducerdesc	transducer type description [char*]
accouple	AC coupling active flag where T=TRUE, F=FALSE [char]
accouplef	AC coupling frequency in Hz [float]
overload:	channel overloaded flag from vxi cards where T=TRUE, F=FALSE [char] [channelsar1]
underrange:	channel under range flag from vxi cards where T=TRUE, F=FALSE [char] [channelsar1]
scalefactor	conversion factor for A/D from volts to EU [float] [channelsar1]
eupervolt	sensor manufacturer calibration factor before preamp from volt to EU [float] [channelsar1]
eupervoltf	sensor manufacturer calibration frequency in Hz [float] [channelsar1]
eupervoltamp	sensor manufacturer calibration amplifier gain [float] [channelsar1]
eucomment	descriptive comment on engineering units used [char*]
chgoodcom	chgood descriptive comment [char*]
chgood	vxi channel good flag where 0=BAD, 1=GOOD [int] [channelsar1]
vxichasslotchan	vxi channel location in chassis/slot/channel combination where = chassis number*10000 + slot number * 100 + channel number [int] [channelsar1]
preampgain	preamplifier gain setting [float] [channelsar1]
powergain	power amplifier gain setting [float] [channelsar1]
extfiltenabled	external filter enabled flag for each channel where 1=TRUE, 0=FALSE [char*]
extfiltlow	external filter low pass frequency in Hz [float] [channelsar1]
extfilthigh	external filter high pass frequency in Hz [float] [channelsar1]
extfiltgain	external filter gain setting [float] [channelsar1]
voltrange	channel voltage range set by user in volts [float] [channelsar1]
voltspercount	conversion factor for A/D from counts to volts [float] [channelar1]

Global Attributes:

title	test title [char*]
datetime	data acquisition date and time [char*]
fileversion	NetCDF file version [char*]
numarrays	number of array system [int]
data_quality_comment	comment for specific run and point [char*]
flow_model_comment	comment on tunnel flow conditions [char*]
rawth_file	raw data source file [char*]

APPENDIX 6. PROCESSED DATA FORMAT DESCRIPTION

The variables and attributes for the processed netCDF data file are described in this appendix. In actuality the processed data file contains all variables and attributes of the raw data file with no dataar1 data. Thus this appendix will describe only the additional variables and attributes added by the processing.

dimensions:

indfdim	number of frequencies contained in the individual channel spectrum variable, set to 1/2 the FFT block size [int]
coord5=5	number of columns in nbspect variable [int]
procmicsdim	number of microphones processed [ind]
fnbpdbdim	number of frequencies processed [int]
xsgdim	number of grid points in the x direction of the scan surface [int]
ysgdim	number of grid points in the y direction of the scan surface [int]
zsgdim	number of grid points in the z direction of the scan surface [int]
timeblksdim	number of FFT blocks requested to average [int]
indcolsdim	number of columns in the ind variable, procmicsdim+1 [int]

variables:

version	processed data version number [int]
procsettup	variable whose attributes are the complete processing setup information [int]
long_name	long name for procsettup [char*]
mapps_con_number	mapps_con processing code version number [char*]
mapps_input_number	mapps_input processing code version number [char*]
mapps_proc_number	mapps_proc processing code version number [char*]
mapps_out_number	mapps_out processing code version number [char*]
fft_block_size	FFT block size [int]
freq_res	hertz per FFT line [float]
freq_up_lim	upper frequency used in delta dB microphone health check and added gains calculation [float]
freq_low_lim	lower frequency used in delta dB microphone health check and added gains calculation [float]
delta_db	delta dB value used in delta dB microphone health check and added gains calculation [float]
req_fft_blkspergroup	number of FFT blocks per group requested to be processed [int]
req_fft_group	number of FFT groups to be processed [int]
num_freqs	number of frequencies to be scanned [int]
antenna_gains	gain to be applied to all channels if use_antenna_gains set to “use_value” [float]
installation_gains	gain in dB to be applied to correct for installation effects, set to zero to disable [float]

windowing_factor set to 1.0 [float]

bad_mics list of user defined bad channels [int] [user defined]

freq_index_nums list of frequencies by index to be processed [int] [fnbpdbdim]

time_stamp date of processing [char*]

proc_type processing type selection (‘regular’, minimum_variance’ or ‘music’) [char*]

noise_reduction noise reduction selection (‘none’, ‘zero_sub’, ‘avg_sub’ or ‘sub_3’) [char*]

side_reduction switch to apply side lob reduction (‘yes’ or ‘no’) [char*]

conv_corr switch to apply flow convection correction (‘yes’ or ‘no’) [char*]

add_gains switch to calculate and apply added gains (‘yes’ or ‘no’) [char*]

windowing_name time history window to apply before calculating FFT (‘rectangular’, ‘tapered’, ‘triangular’, ‘hanning’, ‘hamming’, ‘blackman’, ‘riesz’, ‘riemann’, ‘cauchy’, ‘poisson’ or ‘gaussian’) [char*]

use_antenna_gains switch to use gains stored in raw data file or value specified in antenna_gains attribute (‘use_gains_from_file’ or ‘use_value’) [char*]

input_freq string of requested scan frequencies input line[char*]

output_freq string of actual scan frequencies line [char*]

data_source source of raw time history data (‘NetCDF_file’ or ‘Boeing file’) [char*]

interactive switch for interactive processing (always set to ‘yes’) [char*]

proc_geom switch to choose processing geometry (‘spherical’ or ‘plane_wave’) [char*]

surface switch to determine method of scan surface input for spherical processing (‘define_plane’ or ‘scan_surface_file’) [char*]

surface_file input scan surface file if surface defined as ‘scan_surface_file’ [char*]

phi_format phi format definition (min, max, delta) for plane wave processing [char*]

phi phi geometry definition for plane wave processing [float] [coords]

psi_format psi format definition (min, max, delta) for plane wave processing [char*]

psi psi geometry definition for plane wave processing [float] [coords]

upper_left_format upper left scan surface point format (x, y, z) in inches in model coordinate system [char*]

upper_left upper left scan surface point coordinates in inches in model coordinate system [float] [coord]

lower_left_format lower left scan surface point format (x, y, z) in inches in model coordinate system [char*]

lower_left lower left scan surface point coordinates in inches in model coordinate system [float] [coord]

upper_right_format upper right scan surface point format (x, y, z) in inches in model coordinate system [char*]

upper_right upper right scan surface point coordinates in inches in model coordinate system [float] [coord] num_across number of equally spaced grid points from the upper left to the upper right direction for the ‘define_plane’ option [int]

num_down number of equally spaced grid points from the upper left to the lower left direction for the ‘define_plane’ option [int]

atmos_atten switch to activate the atmospheric attenuation correction (‘on’ or ‘off’) [char*]

directivity switch to activate the array directivity correction (‘on’ or ‘off’) [char*]

directivity_file file containing the directivity correction data for directivity set to ‘on’ [char*]

density switch to activate the density correction (‘on’ or ‘off’) [char*]

freefield_cal switch to activate microphone free-field effect correction (‘on’ or ‘off’) [char*]

cal_type calibration type indicator (‘basic’, ‘ames’ or ‘boeing’) [char*]

acc_fit_type type of curve fit used in speaker calibration [char*]

speaker_name name of speaker used for calibration [char*]

speaker_cal_file file containing speaker calibration curve fit [char*]

ncfile netCDF file containing instrument calibrations [char*]

diaphragm_ser switch to activate the diaphragm serial number based electrostatic frequency correction (‘on’ or ‘off’) [char*]

diaphragm_rep switch to activate the diaphragm model number based electrostatic frequency correction (‘on’ or ‘off’) [char*]

diaphragm_cal flag to determine what part of the diaphragm electrostatic calibration to apply (‘amplitude’, ‘phase’ or ‘both’) [char*]

preamp_ser switch to activate the preamp serial number based electrostatic frequency correction (‘on’ or ‘off’) [char*]

preamp_rep switch to activate the preamp model number based electrostatic frequency correction (‘on’ or ‘off’) [char*]

preamp_cal flag to determine what part of the preamp electrostatic calibration to apply (‘amplitude’, ‘phase’ or ‘both’) [char*]

power_ser switch to activate the power supply serial number based electrostatic frequency correction (‘on’ or ‘off’) [char*]

power_rep switch to activate the power supply model number based electrostatic frequency correction (‘on’ or ‘off’) [char*]

power_cal flag to determine what part of the power supply electrostatic calibration to apply (‘amplitude’, ‘phase’ or ‘both’) [char*]

filter_ser switch to activate the filter serial number based electrostatic frequency correction (‘on’ or ‘off’) [char*]

filter_rep switch to activate the filter model number based electrostatic frequency correction (‘on’ or ‘off’) [char*]

filter_cal flag to determine what part of the filter electrostatic calibration to apply (‘amplitude’, ‘phase’ or ‘both’) [char*]

wire_chan switch to activate the institutional wiring channel based electrostatic frequency correction (‘on’ or ‘off’) [char*]

wire_rep switch to activate the institutional wiring representative based electrostatic frequency correction (‘on’ or ‘off’) [char*] wire_cal flag to determine what part of the institutional wiring electrostatic calibration to apply (‘amplitude’, ‘phase’ or ‘both’) [char*]

delta_pres_val switch to activate the delta pressure from atmospheric base on the diaphragm serial number frequency correction (‘on’ or ‘off’) [char*]

delta_pres_rep switch to activate the delta pressure from atmospheric base on the diaphragm model number frequency correction (‘on’ or ‘off’) [char*]

delta_pres_cal flag to determine what part of the delta pressure calibration to apply (‘amplitude’, ‘phase’ or ‘both’) [char*]

sensitivity_file file containing the individual channel pascals per volt sensitivity [char*]

sensitivity_source source of the sensitivity calibration values [char*]

instr_cal_file input netCDF file containing the instrumentation calibration neural net curve fits [char*]

installation_gain_type flag to indicate if installation gains are from file or a constant [char*]

installation_gain_file input netCDF file containing the installation calibration neural net curve fits [char*]

ctrl_proc processor name on which to run the control process [char*]

ctrl_exec executable path for the control process [char*]

raw_proc processor name on which to run the read raw data process [char*]

raw_exec executable path for the read raw data process [char*]

out_proc processor name on which to run the output process [char*]

out_exec executable path for the output process [char*]

out_path processed data output file name including path [char*]

comp_proc processor names on which the scanning processes will be run [char*]

comp_exec executable paths for the scanning processes [char*]

num_cpu number of CPUs to use on each comp_proc [int] [number of machines]

lead_cpu number of the lead processor [int]

user_conditions_comment comment about the wind tunnel conditions [char*]

wt_conditions_source source for the wind tunnel conditions to be used in processing (‘netCDF_raw_file’ or ‘user_input’) [char*]

mach_number wind tunnel Mach number [float]

static_temp_rankine user supplied tunnel static temperature in °R [float]

static_pres_psf user supplied tunnel static pressure in psf [float]

rel_humidity user supplied relative humidity in percent [float]

num_channels number of channels in input data [int]

num_good_mics number of good microphones in processed data [int]

num_blocks number of requested FFT blocks to process [int]

num_good_blocks number of good FFT blocks in processed data [int]

mic_status indicator of microphone health status [int][channelsar1]

slope sensitivity value used in processing [float] [channelsar1]

hosts_par_f file name containing a list of processors to be used in the NAS system [char*]
data_tmp_f temporary debugging file written by the input program [char*]
raw_time_f raw time history file name [char*]
file_log_f log file [char*]

nbspect narrow band spectral results containing the average spectrum from all the microphones before scanning, the mean and the maximum value for all the scan points for each frequency processing in scanning with the column order being fin, f(Hz), asp(dB), mea(dB), max(dB) [float] [fnbpdbdim, coord5]

long_name long name for nbspect variable [char*]
format variable format [char*]
units nbspect units [char*]

mhlth array of characters indicating the individual microphone and block health with the time blocks along columns and channels along rows [char] [channelsar1, timeblksdim]

long_name mhlth variable long name [char*]
format the variable format [char*]
options character options are 0=good mic, 1=band edged, 2=flat spot, 3=delta db, 4=declared bad, 5=not used [char*]

micblkhlth array of characters indicating the actual data processed including rows and columns of data discarded due to microphone health check failures [char] [channelsar1, timeblksdim]

long_name micblkhlth variable long name [char*]
format the variable format [char*]
options character options are 0=good mic, 1=band edged, 2=flat spot, 3=delta db, 4=declared bad, 5=not used, 6=bad mic flag, 7=bad blocks, 9=other [char*]
micflg_options microphone options are 0 = bad mic and 1 = good mic
micflg integer indicator if microphone was used in processing [int][channelsar1]
blkflg_options FFT block options are 0 = bad block and 1 = good block
blkflg inteeger indicator if FFT block was used in processing [int][timeblksdim]

ind individual microphone spectra which have been averaged over FFT blocks. First column is the frequency with the individual channel spectra following. [float] [indfdim, indcolsdim]

long_name ind long name [char*]
format the variable format [char*]
units ind units [char*]

sgoutx [float] [xsgdim, ysgdim, zsgdim]

long_name sgoutx variable long name [char*]
units sgoutx units [char*]
format the variable format [char*]
xdim number of the x values [int]

ydim	number of y values [int]
zdim	number of z values [int]
sgouty	[float] [xsgdim, ysgdim, zsgdim]
long_name	sgouty variable long name [char*]
units	sgouty units [char*]
format	the variable format [char*]
xdim	number of the x values [int]
ydim	number of y values [int]
zdim	number of z values [int]
sgoutz	[float] [xsgdim, ysgdim, zsgdim]
long_name	sgoutz variable long name [char*]
units	sgoutz units [char*]
format	the variable format [char*]
xdim	number of the x values [int]
ydim	number of y values [int]
zdim	number of z values [int]
nbpdb	[float] [xsgdim, ysgdim, zsgdim, fnbpdbdim]
long_name	nbpdb variable long name [char*]
units	nbpdb units [char*]
format	the variable format [char*]
xdim	number of the x values [int]
ydim	number of y values [int]
zdim	number of z values [int]
fdim	number of frequency values [int]

Global Attributes:

process_out_format_version	version number of the processed data format [char*]

APPENDIX 7. CONTROL SETTINGS FILE EXAMPLE

```
$MAIN_NUM
 version_number = 10,
 fft_block_size = 512,
 freq_res = 150.000000,
 req_fft_blkspergroup = 80,
 req_fft_group = 5,
 num_freqs = 166,
 antenna_gains = 20.000000,
 installation_gains = 6.0,
 windowing_factor = 1.0,
 bad_mics = ,
 freq_index_nums = 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
                      26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
                      45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,
                      64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
                      83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100,
                      101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114,
                      115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128,
                      129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
                      143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
                      157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168,
 freq_up_lim = 10000.000000,
 freq_low_lim = 1000.000000,
 delta_db = 6.000000,
 $
 $MAIN_STR
 time_stamp = ‘Thu Apr 9 11:13:41 1998’,
 proc_type = ‘regular’,
 noise_reduction = ‘sub_3’,
 side_reduction = ‘yes’,
 conv_corr = ‘yes’,
 add_gains = ‘no’,
 windowing_name = ‘hanning’,
 use_antenna_gains = ‘use_gains_from_file’,
 input_freq = ‘300(1)25000’,
 output_freq = ‘300.00(1)25050.00’,
 data_source = ‘NetCDF_file’,
 interactive = ‘yes’,
 $
 $GEOM_STR
 proc_geom = ‘spherical’,
 surface = ‘define_plane’,
 surface_file = ‘/data1/bardina/results/geom/scanV01.bin’,
 $
```

```
$GEOM_NUM
phi_min = -90.000000,
phi_max = 90.000000,
phi_delta = 1.000000,
psi_min = -90.000000,
psi_max = 90.000000,
psi_delta = 1.000000,
upper_left_x = 30.000000,
upper_left_y = 60.000000,
upper_left_z = -5.000000,
lower_left_x = 30.000000,
lower_left_y = 0.000000,
lower_left_z = -5.000000,
upper_right_x = -30.000000,
upper_right_y = 60.000000,
upper_right_z = -5.000000,
num_across = 121,
num_down = 121,
$
$CAL_STR
atmos_atten = ‘off’,
directivity = ‘off’,
directivity_file = ‘/pmat1d2/data/fe3/test/directivity.nc’,
density = ‘off’,
freefield_cal = ‘off’,
cal_type = ‘basic’,

diaphragm_ser = ‘off’,
diaphragm_rep = ‘off’,
diaphragm_cal = ‘both’,
preamp_ser = ‘off’,
preamp_rep = ‘off’,
preamp_cal = ‘both’,
power_ser = ‘off’,
power_rep = ‘off’,
power_cal = ‘both’,
filter_ser = ‘off’,
filter_rep = ‘off’,
filter_cal = ‘both’,
wire_chan = ‘off’,
wire_rep = ‘off’,
wire_cal = ‘both’,
delta_pres_val = ‘off’,
delta_pres_rep = ‘off’,
delta_pres_cal = ‘both’,
sensitivity_file = ‘/pmat1d2/data/fe3/test/piston.cal’,
sensitivity_source = ‘pistonphone’,
 instr_cal_file = ‘/pmat1d2/data/fe3/calibration980421.nc/’,
 installation_gain_type = ‘contant’,
 installation_gain_file = ‘/pmat1d2/data/fe3/calibration980421.nc/’,
 $
 $SETUP_STR
 ctrl_proc = ‘pmat1’,
 ctrl_exec = ‘/usr/local/pvm3/bin/SGI64/mapps_con’,
 raw_proc = ‘pmat1’,
 raw_exec = ‘/usr/local/pvm3/bin/SGI64/mapps_input’,
 out_proc = ‘pmat1’,
 out_exec = ‘/usr/local/pvm3/bin/SGI64/mapps_out’,
 out_path = ‘/pmat1d2/data/fe3/test/105000PMA00014010procd10.nc’,
 comp_proc(1) = ‘leonardo’,
 comp_proc(2) = ‘proc2’,
 comp_proc(3) = ‘proc3’,
 comp_proc(4) = ‘proc4’,
 comp_proc(5) = ‘proc5’,
 comp_proc(6) = ‘proc6’,
 comp_proc(7) = ‘proc7’,
 comp_proc(8) = ‘proc8’,
 comp_proc(9) = ‘proc9’,
 comp_proc(10) = ‘proc10’,
 comp_proc(11) = ‘proc11’,
 comp_proc(12) = ‘proc12’,
 comp_proc(13) = ‘proc13’,
 comp_proc(14) = ‘proc14’,
 comp_proc(15) = ‘proc15’,
 comp_proc(16) = ‘proc16’,
 comp_exec(1) = ‘/usr/local/pvm3/bin/SGI64/mapps_proc’,
 comp_exec(2) = ‘exec2’,
 comp_exec(3) = ‘exec3’,
 comp_exec(4) = ‘exec4’,
 comp_exec(5) = ‘exec5’,
 comp_exec(6) = ‘exec6’,
 comp_exec(7) = ‘exec7’,
 comp_exec(8) = ‘exec8’,
 comp_exec(9) = ‘exec9’,
 comp_exec(10) = ‘exec10’,
 comp_exec(11) = ‘exec11’,
 comp_exec(12) = ‘exec12’,
 comp_exec(13) = ‘exec13’,
 comp_exec(14) = ‘exec14’,
 comp_exec(15) = ‘exec15’,
 comp_exec(16) = ‘exec16’,
 $
 $SETUP_NUM
iprocess = 1,
idebug = 0,
num_cpu(1) = 8,
num_cpu(2) = 0,
num_cpu(3) = 0,
num_cpu(4) = 0,
num_cpu(5) = 0,
num_cpu(6) = 0,
num_cpu(7) = 0,
num_cpu(8) = 0,
num_cpu(9) = 0,
num_cpu(10) = 0,
num_cpu(11) = 0,
num_cpu(12) = 0,
num_cpu(13) = 0,
num_cpu(14) = 0,
num_cpu(15) = 0,
num_cpu(16) = 0,
lead_cpu = 1,
 $
 $CONDITIONS_STR
user_conditions_comment = ‘none’,
wt_conditions_source = ‘netCDF_raw_file’,
 $
 $CONDITIONS_NUM
mach_number = 0.219000,
static_temp_rankine = 526.148010,
static_pres_psf = 2059.996094,
rel_humidity = 46.400002,
 $
$ARRAY_NUM
num_channels = 100,
num_good_mics = 100,
mic_status = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0,
 $
 $OLD_STUFF
hosts_par_f = ‘/tmp/105000PMA00014010direc10.par’,
data_tmp_f = ‘/tmp/105000PMA00014010headr10.nc’,
raw_time_f = ‘/pmat1d2/data/fe3/test/105000PMA00014010rawth99.nc’,
file_log_f = ‘/tmp/105000PMA00014010direc10.log’,
 $
```

APPENDIX 8. INSTRUMENTATION CALIBRATION FILE DESCRIPTION

The Instumentation Calibration File is a netCDF data file. No dimensions are used. All of the data are stored in attributes. Variables are indicated in bold type with their attributes in normal type.

variables:

general	empty [float]
desc	information [char*]
microphone	empty [float]
desc1	information [char*]
.	.
.	.
desc66	information [char*]
preamp	empty [float]
desc1	information [char*]
.	.
.	.
desc9	information [char*]
powersuply	empty [float]
desc1	information [char*]
.	.
.	.
desc9	information [char*]
filter	empty [float]
desc1	information [char*]
.	.
.	.
desc9	information [char*]
wire	empty [float]
desc1	information [char*]
.	.
.	.
desc9	information [char*]
micsn7193	empty (microphone serial number equals 7193 in this example) [char]
caldatafile	name of the instrumentation calibration file [char*]
instrument	type of instrument [char]
sn	serial number of instrument [char]
brand	brand of instrument [char
model	model of instrument [char]
size	size of instrument [char]
calvolts	calibration voltage level [float]
caldb	calibration dB level [float]

calfreq	calibration frequency [float]
powsupsn	power supply serial number [char*]
powsupbrand	power supply brand [char*]
powsupmodel	power supply model [char*]
psstatus	power supply status [char*] 1 for power supply in model, 0 for no power supply, –1 for effect of power supply removed by model of power supply, –2 for effect of power supply removed by model of power supply instrument type
preampsn	preamp serial number [char*]
preampbrand	preamp brand [char*]
preampmodel	preamp model [char*]
preampstatus	preamp status [char*] 1 for preamp in model, 0 for no preamp, –1 for effect of preamp removed by model of preamp, –2 for effect of preamp removed by model of preamp instrument type
diaphragmsn	diaphragm serial number [char*]
diaphragmbrand	diaphragm brand [char*]
diaphragmmodel	diaphragm model [char*]
diaphstatus	diaphragm status [char*] 1 for diaphragm in model, 0 for no diaphragm, –1 for effect of diaphragm removed by model of diaphragm, –2 for effect of diaphragm removed by model of diaphragm instrument type
filtersn	filter serial number [char*]
filterbrand	filter brand [char*]
filtermodel	filter model [char*]
filtstatus	filter status [char*] 1 for filter in model, 0 for no filter, –1 for effect of filter removed by model of filter, –2 for effect of filter removed by model of filter instrument type
wiresn	wire serial number [char*]
wirestatus	wire status [char*] 1 for wire in model, 0 for no wire, –1 for effect of wire removed by model of wire, –2 for effect of wire removed by model of wire instrument type
domodel	vector describing which models were made [float] domodel(1) = 1 for amplitude model domodel(2) = 1 for phase model domodel(3) = 1 for delta amplitude model domodel(4) = 1 for delta phase model

ampcaldate	date instrument was calibrated [char*]
ampdate	date amplitude model was made [char*]
ampquality	quality of amplitude model [float]
ampreal	indicates if real data was used [float]
ampstats	statistics of data used to generate the neural net model; ampstats (1) is the minimum wave number (1/m) for diaphram or minimum frequency (Hz) for other instruments, ampstats (2) is the maximum wave number (1/m) for diaphram or maximun frequency (Hz) for other instruments, ampstats (3) is the average wave number (1/m) for diaphram or average frequency (Hz) for other instruments, ampstats (4) is the standard deviation of wave number (1/m) for diaphram or standard deviation of frequency (Hz) for other instruments, ampstats (5) is minimum pressure (psf), ampstats (6) is maximum pressure (psf), ampstats (7) is mean pressure (psf), ampstats (8) is standard deviation of pressure (psf)[float]
ampnumneurons	number of neurons [float]
ampinvars	number of input variables to model [float]
ampfun1	1st function for model [char*]
ampw1	matrix of w1 weights [float]
ampb1	vector of b1 biases [float]
ampfun2	2nd function for model [char*]
ampw2	matrix of w2 weights (vector in this case) [float]
ampb2	vector of b2 bias (length 1 in this case) [float]
ampcorr	correlation coefficient [float]
amprms	normalized rms error [float]
amppeak	peak error in dB [float]
phasecaldate	date instrument was calibrated [char*]
.	repeat information for phae
.	
phasepeak	peak error in dB [float]
delampcaldate	date instrument was calibrated [char*]
.	repeat information for delta amplitude
.	
delamppeak	peak error in dB [float]
delphasecaldate	date instrument was calibrated [char*]
.	repeat informatin for delta phase
.	
delphasepeak	peak error in dB [float]
powsupsn7500.A	empty (power supply serial number equals 7500.A in this example) [char]
caldatafile	name of the instrumentation calibration file [char*]
.	repeat information for power supply
.	
delphasepeak	peak error in dB [float]
preampsn7311	empty (preamplifier serial number equals 7311 in this example) [char]

caldatafile	name of the instrumentation calibration file [char*]
.	repeat information for power supply
.	
phasepeak	peak error in dB [float]
filtersn123	empty (filter serial number equals 123 in this example) [char]
caldatafile	name of the instrumentation calibration file [char*]
.	repeat information for power supply
.	
phasepeak	peak error in dB [float]
wiresn456	empty (wire serial number equals 456 in this example) [char]
caldatafile	name of the instrumentation calibration file [char*]
.	repeat information for power supply
.	
phasepeak	peak error in dB [float]
micsnrepTMS140BF	empty (microphone model equals TMS140BF in this example) [char]
caldatafile	name of the instrumentation calibration file [char*]
.	repeat information for power supply
.	
phasepeak	peak error in dB [float]
preampsnrepTMS112AA	empty (preamplifier model equals TNS112AA in this example) [char]
caldatafile	name of the instrumentation calibration file [char*]
.	repeat information for power supply
.	
phasepeak	peak error in dB [float]
preampsnrepTMS126AC	empty (preamplifier model equals TMS126AC in this example) [char]
caldatafile	name of the instrumentation calibration file [char*]
.	repeat information for power supply
.	
phasepeak	peak error in dB [float]
filtersnrepABC	empty (filter model equals ABC in this example) [char]
caldatafile	name of the instrumentation calibration file [char*]
.	repeat information for power supply
.	
phasepeak	peak error in dB [float]
wiresnrepDEF	empty (wire model equals DEF in this example) [char]
caldatafile	name of the instrumentation calibration file [char*]
.	repeat information for power supply
.	
phasepeak	peak error in dB [float]

APPENDIX 9. SPEAKER CALIBRATION FILE DESCRIPTION

The Speaker Calibration File is a netCDF data file. No dimensions are used. All of the data are stored in attributes. Variables are indicated in bold type with their attributes in normal type.

variables:

general empty [float]

desc information [char*]

speakerdefinition empty [float]

desc1 information [char*]

. .

. .

desc55 information [char*]

speakerSJ1 empty (speaker name equals SJ1 in this example) [char]

speakname name of the speaker [char*]

lowerwaveno lower wave number limit (1/m) [float]

upperwaveno upper wave number limit (1/m) [float]

minfreq lower frequency limit (Hz) [float]

maxfreq upper frequency limit (Hz) [float]

speakdate history of dates of model generation [char*]

refnumneurons number of neurons for reference location [float]

reffun1 1st function for model for reference location [char*]

refw1 matrix of w1 weights (vector in this case) for reference location [float]

reflb1 vector of b1 biases for reference location [float]

reffun2 2nd function for model for reference location [char*]

refw2 matrix of w2 weights (vector in this case) for reference location [float]

refb2 vector of b2 bias (length 1 in this case) for reference location [float]

refcor correlation coefficient for reference location [float]

refrms normalized rms error for reference location [float]

refpeak peak error in dB for reference location [float]

speakquality quality of the speaker model, = 1 for good, = 0 for bad [float]

speakstats statistics of data used to generate the neural net model; speakstats(1) is the minimum wave number (1/m), speakstats(2) is the maximum wave number (1/m), speakstats(3) is the average wave number (1/m), speakstats(4) is the standard deviation of wave number (1/m), speakstats(5) is minimum theta (deg), speakstats(6) is maximum theta (deg), speakstats(7) is mean theta (deg), speakstats(8) is standard deviation of theta (deg), speakstats(9) is minimum phi (deg), speakstats(10) is maximum phi (deg), speakstats(11) is mean phi (deg), speakstats(12) is standard deviation of phi (deg) [float]

speaknumneurons number of neurons [float]

speakinvars	number of variables in speaker model [float]
speakfun1	1st function for model [char*]
speakw1	matrix of w1 weights [float]
speakb1	vector of b1 biases [float]
speakfun2	2nd function for model [char*]
speakw2	matrix of w2 weights (vector in this case) [float]
speakb2	vector of b2 bias (length 1 in this case) [float]
speakcor	correlation coefficient [float]
speakrms	normalized rms error [float]
speakpeak	peak error in dB [float]

APPENDIX 10. INSTALLATION CALIBRATION FILE DESCRIPTION

The Installation Calibration File is a netCDF data file. No dimensions are used. All of the data are stored in attributes except for the one variable “numinstallocations”. Variables are indicated in bold type with their attributes in normal type.

variables:

general	empty [float]
desc	information [char*]
installation	empty [float]
desc1	information [char*]
.	.
.	.
desc31	information [char*]
numinstallocations	number of microphone locations [float]
instalmic1	empty [char]
instalarrayname	name of the array [char*]
instalindex	index number of the location on the array pattern(=1 in this example) [char*]
instalx	x location in inches in array coordinates [float]
instaly	y location in inches in array coordinates [float]
instalz	z location in inches in array coordinates [float] [float]
instalquality	quality of the installation model, = 1 for good, = 0 for bad [float]
instalmicdate	history of dates of model generation [char*]
instalstats	statistics of data used to generate the neural net model; instalstats(1) is the minimum wave number, instalstats(2) is the maximum wave number, instalstats(3) is the average wave number, instalstats(4) is the standard deviation of wave number [float]
instalnumneurons	number of neurons [float]
instalfun1	1st function for model [char*]
instalw1	matrix of w1 weights (vector in this case) [float]
instalb1	vector of b1 biases [float]
instalfun2	2nd function for model [char*]
instalw2	matrix of w2 weights (vector in this case) [float]
instalb2	vector of b2 bias (length 1 in this case) [float]
instalcor	correlation coefficient [float]
instalrms	normalized rms error [float]
instalpeak	peak error in dB [float]
instalmic2	repeat information for each microphone location

APPENDIX 11. DIRECTIVITY CALIBRATION FILE DESCRIPTION

The Directivity Calibration File is a netCDF data file. No dimensions are used. All of the data are stored in attributes . Variables are indicated in bold type with their attributes in normal type.

variables:

general	empty [float]
desc	information [char*]
directivity	empty [float]
desc1	information [char*]
.	.
.	.
desc30	information [char*]
dir710-01-100	empty (array name equals 710-01-100 in this example) [char]
dirarrayname	name of the array [char*]
dirdate	history of dates of model generation [char*]
dirquality	quality of the directivity model, = 1 for good, = 0 for bad [float]
dirstat	statistics of data used to generate the neural net model; dirstat(1) is the minimum wave number(1/m), dirstat(2) is the maximum wave number (1/m), dirstat(3) is the average wave number(1/m), dirstat(4) is the standard deviation of wave number (1/m), dirstat(5) is minimum theta (deg), dirstat(6) is maximum theta (deg), dirstat(7) is mean theta (deg), dirstat(8) is standard deviation of theta (deg), dirstat(9) is minimum phi (deg), dirstat(10) is maximum phi (deg), dirstat(11) is mean phi (deg), dirstat(12) is standard deviation of phi (deg) [float]
dirnumneurons	number of neurons [float]
dirinvars	number of model variables [float]
dirfun1	1st function for model [char*]
dirw1	matrix of w1 weights [float]
dirb1	vector of b1 biases [float]
dirfun2	2nd function for model [char*]
dirw2	matrix of w2 weights (vector in this case) [float]
dirb2	vector of b2 bias (length 1 in this case) [float]
dircor	correlation coefficient [float]
dirrms	normalized rms error [float]
dirpeak	peak error in dB [float]

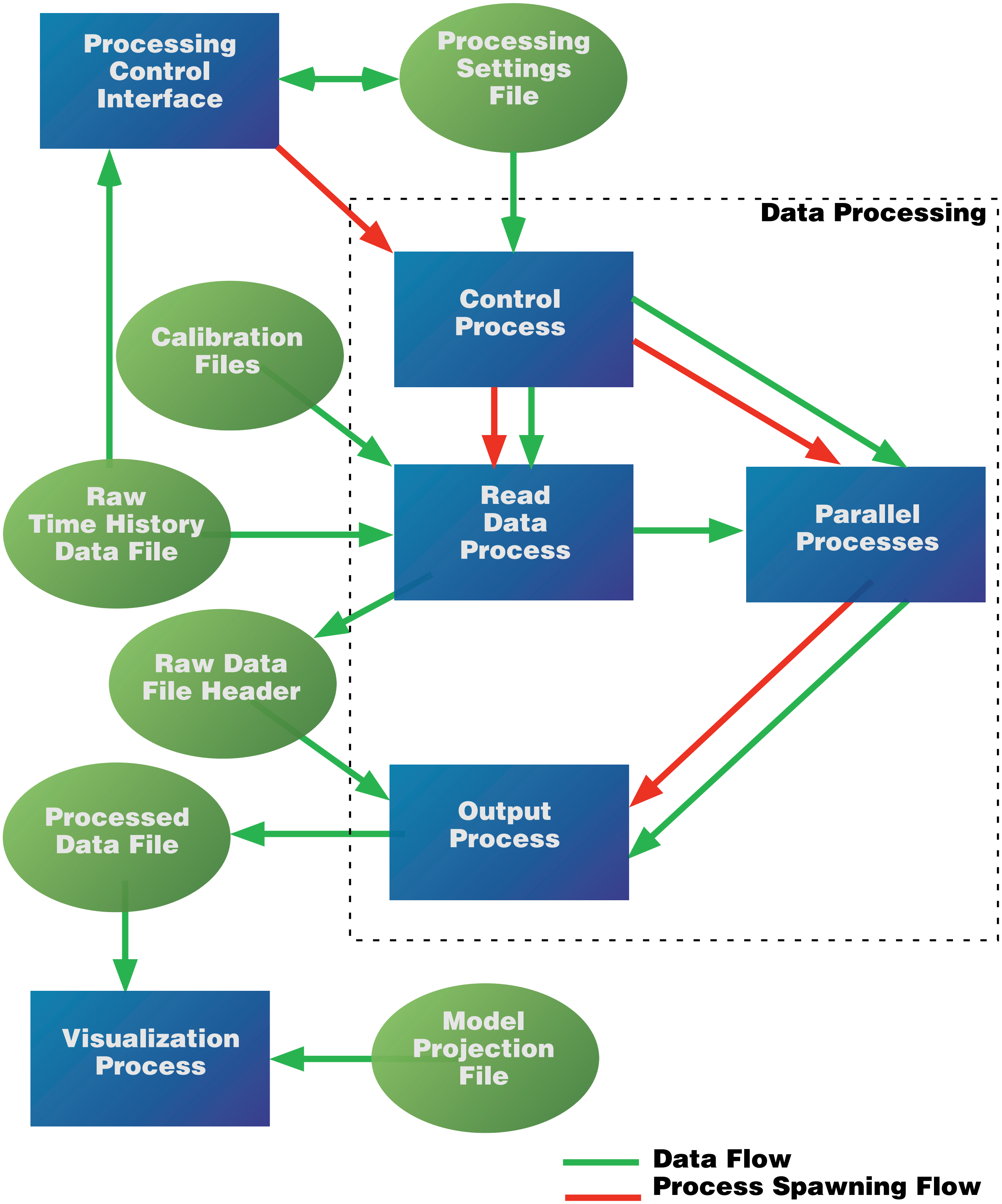


Figure 1. MAPPS overview.

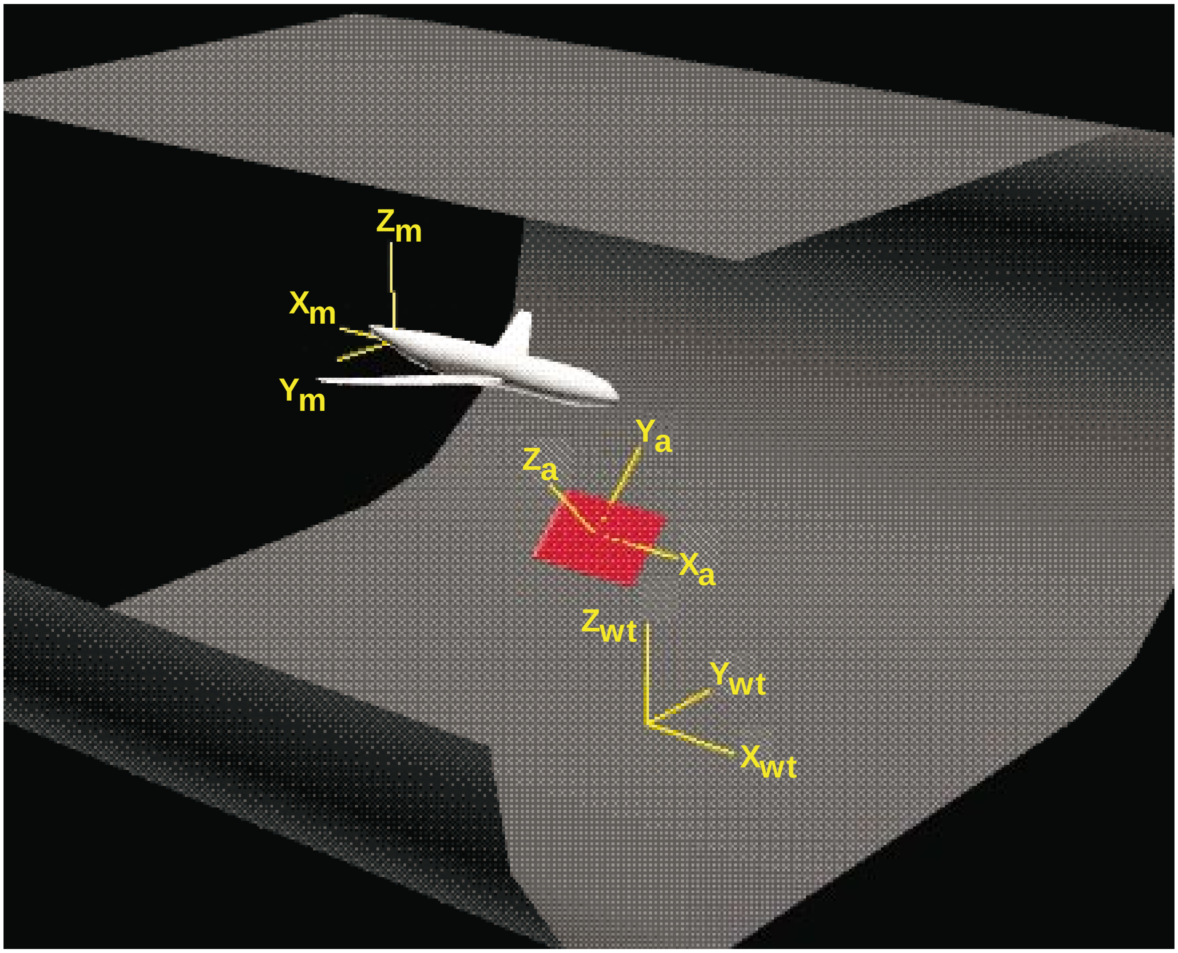
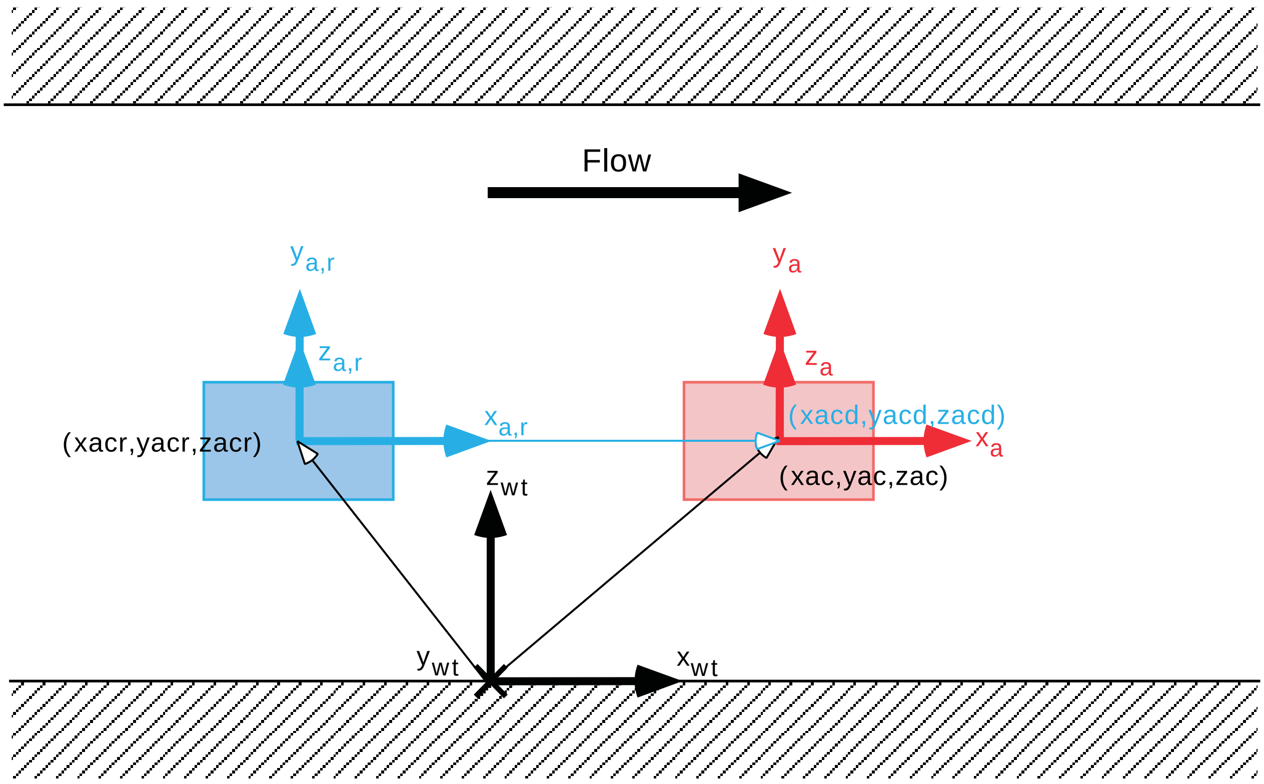
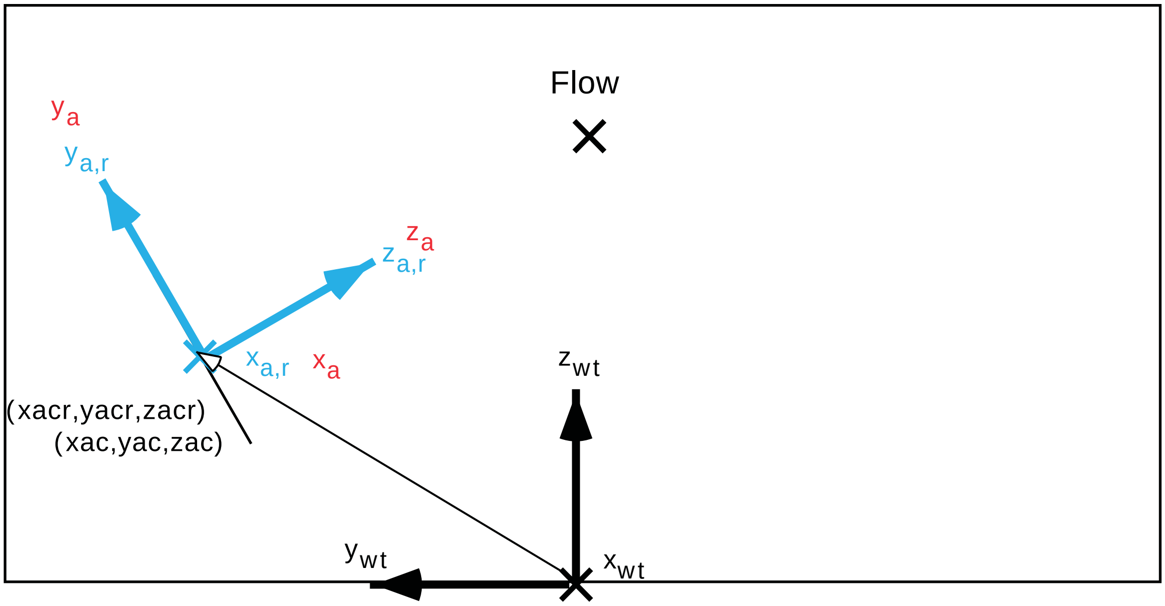


Figure 2. Coordinate systems used in MAPPS.

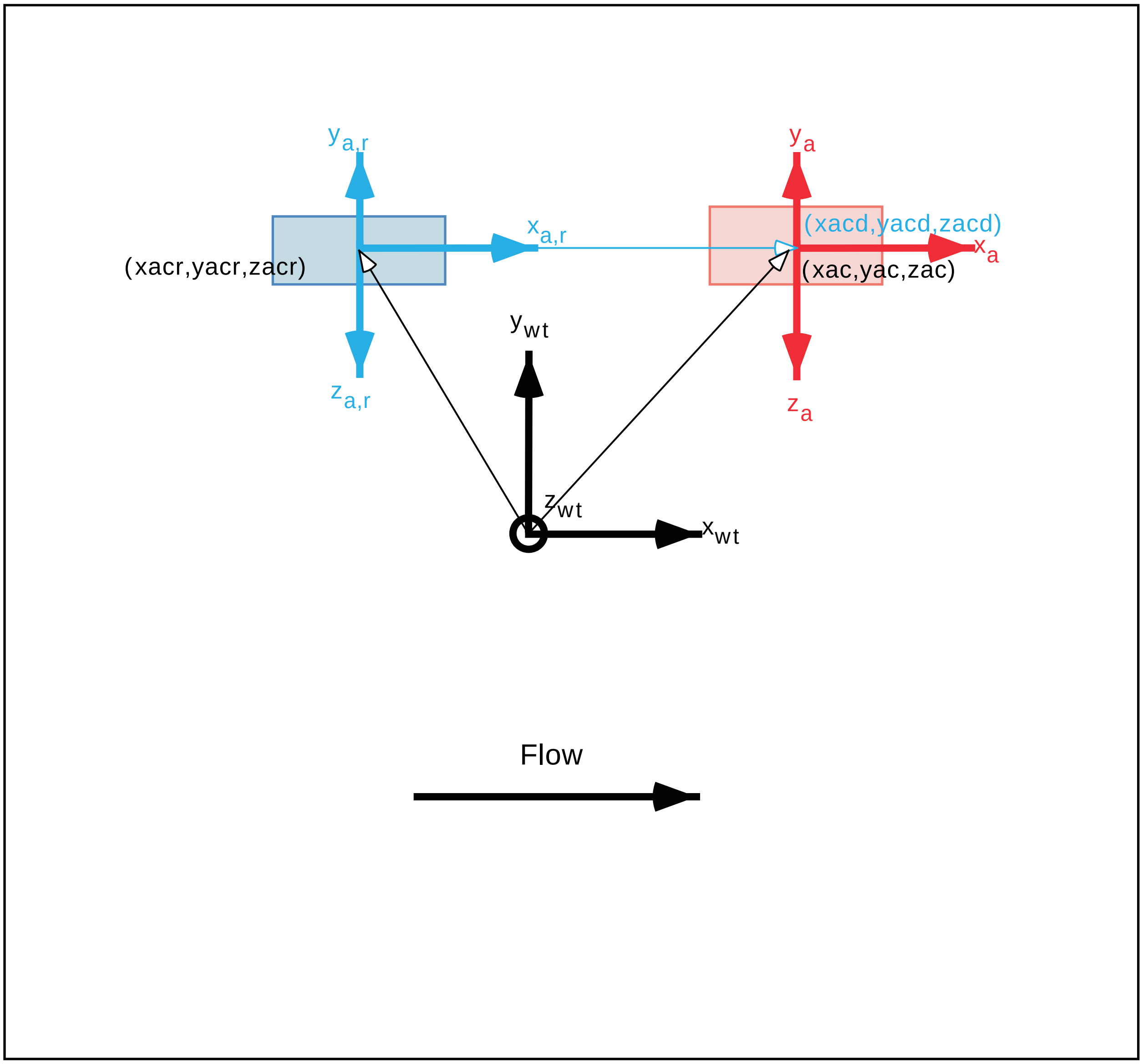


a) Side view of array coordinates



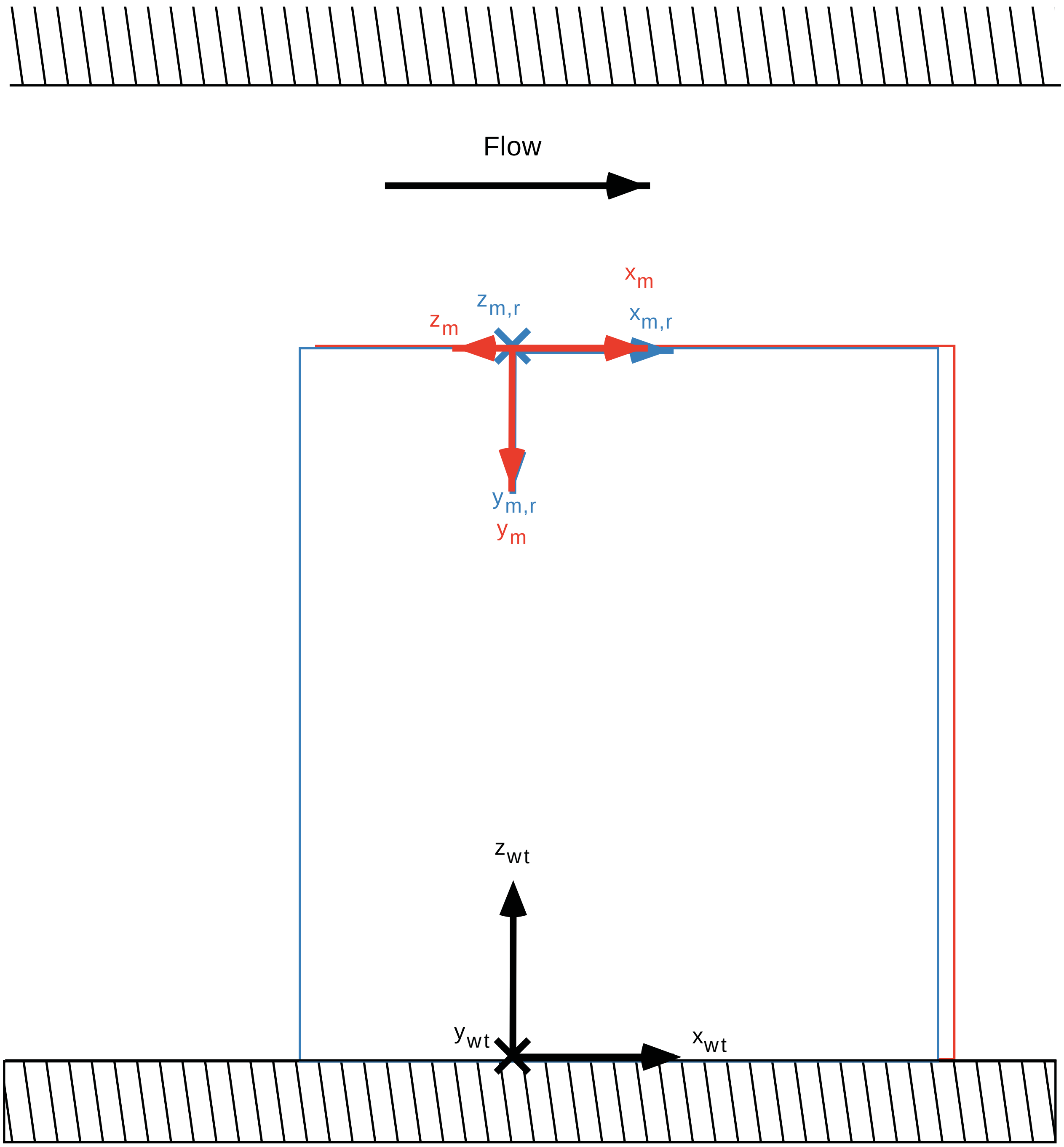
b) End view of array coordinates

Figure 3. View of array coordinates.



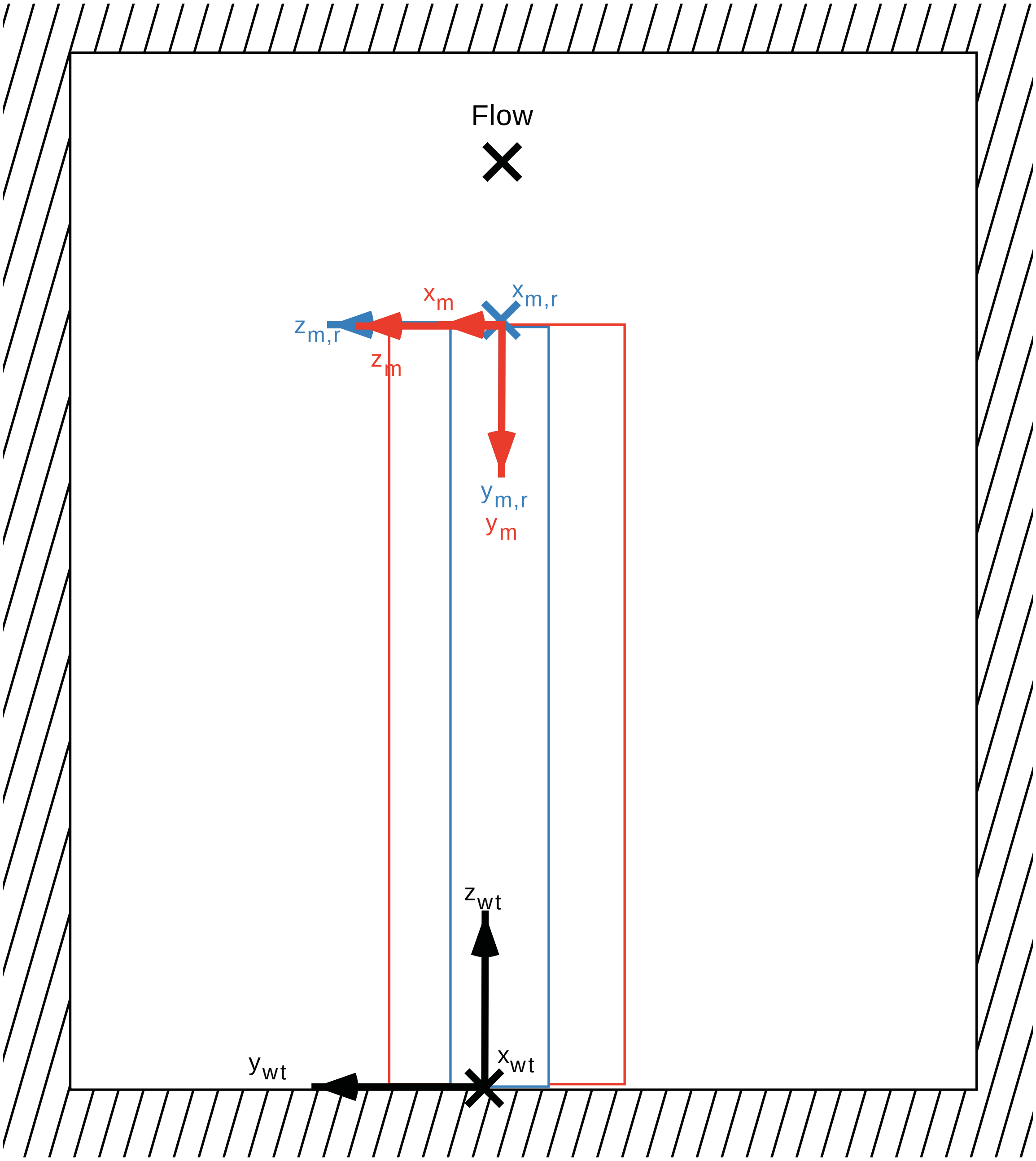
c) Top View of Array Coordinates

Figure 3. View of array coordinates (concluded)



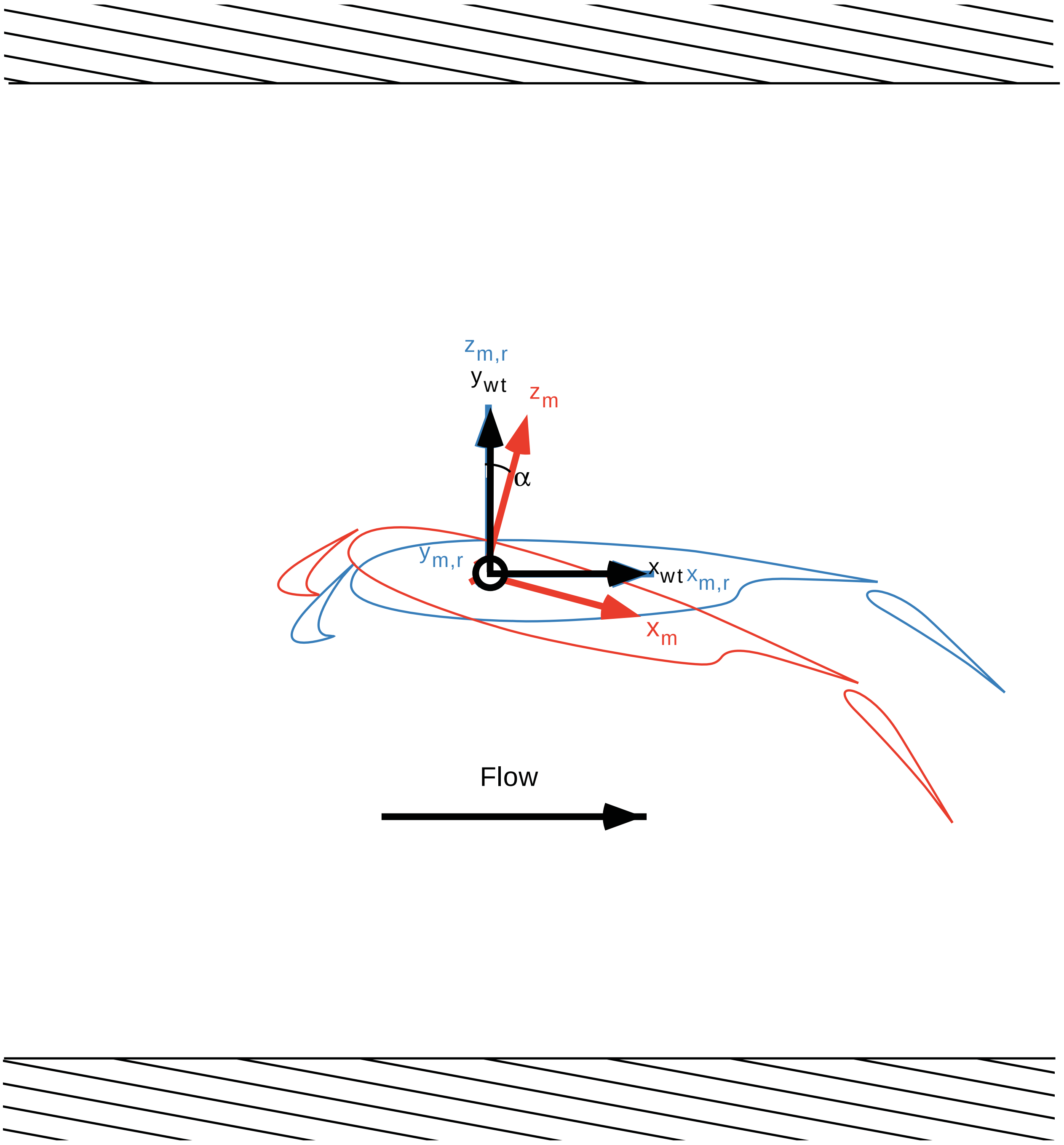
a) Side view of model coordinates

Figure 4. View of model coordinates.



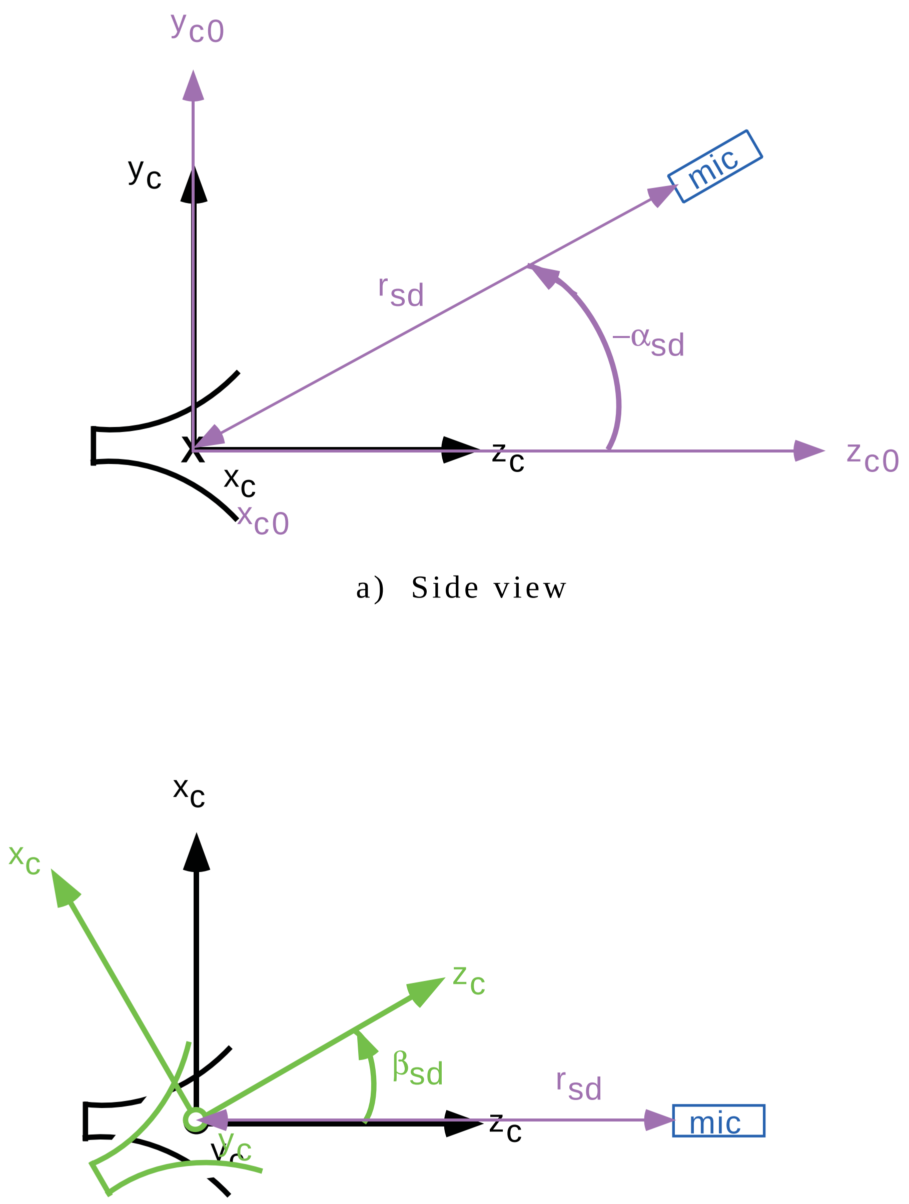
b) End View of Model Coordinates

Figure 4. View of model coordinates (continued).



c) Top view of model coordinates

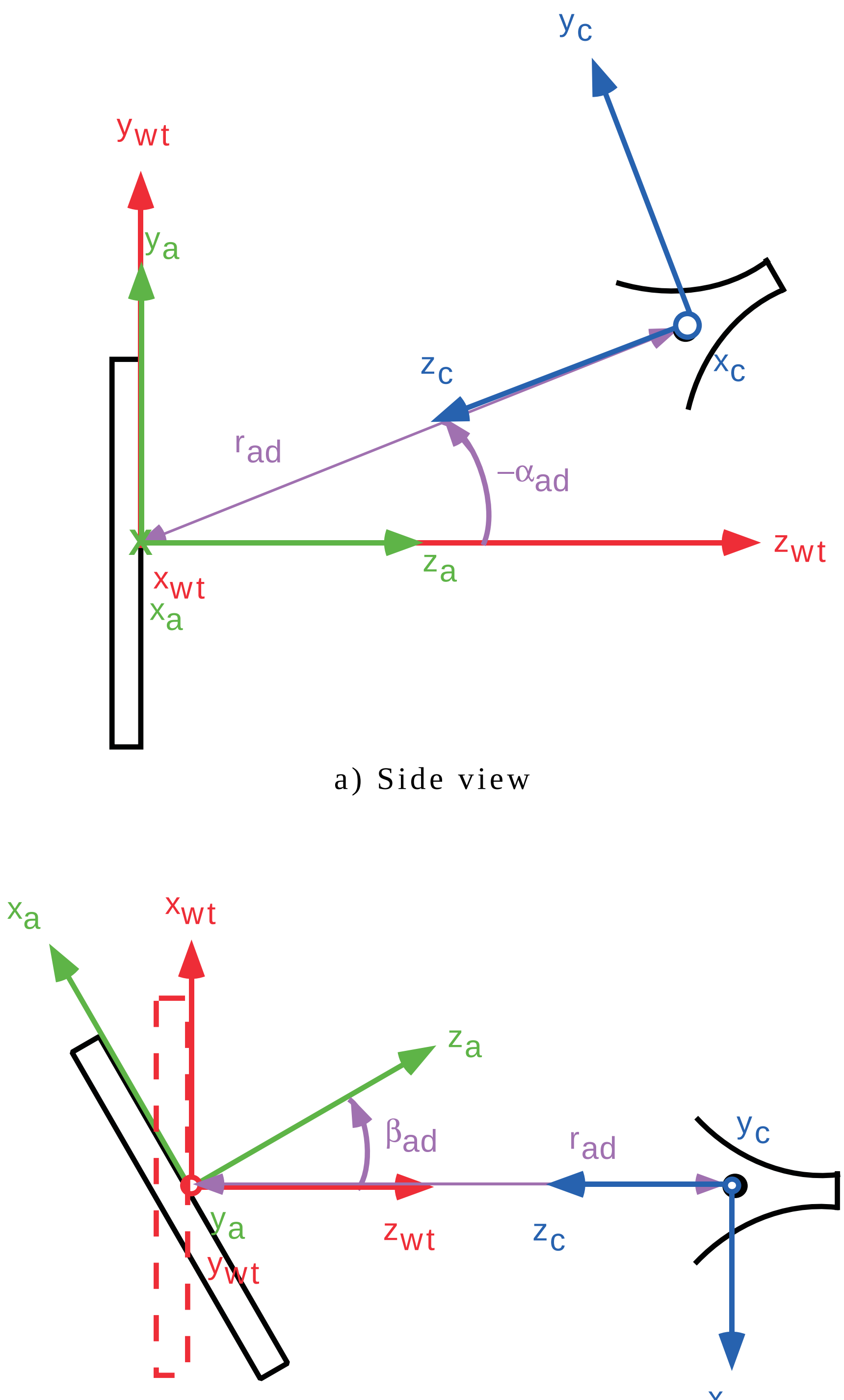
Figure 4. View of model coordinates (concluded).



a) Side view

b) Top view

Figure 5. Speaker definition calibration coordinate system.



a) Side view

b) Top view

Figure 6. Array directivity calibration coordinate systems.

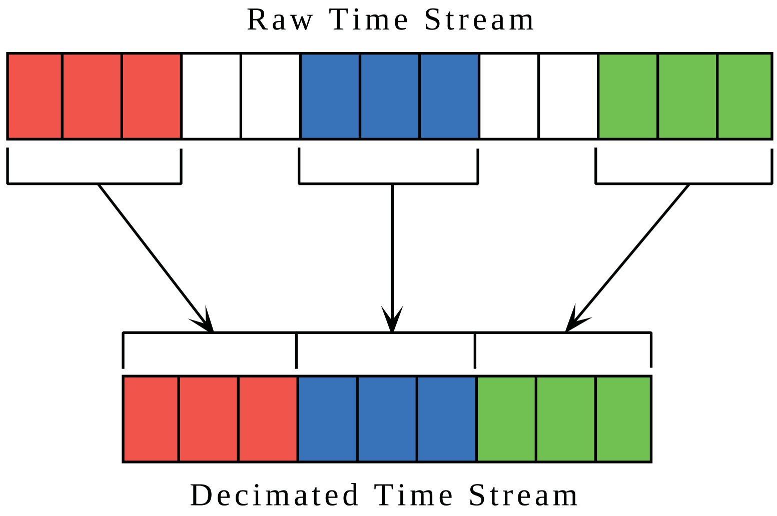
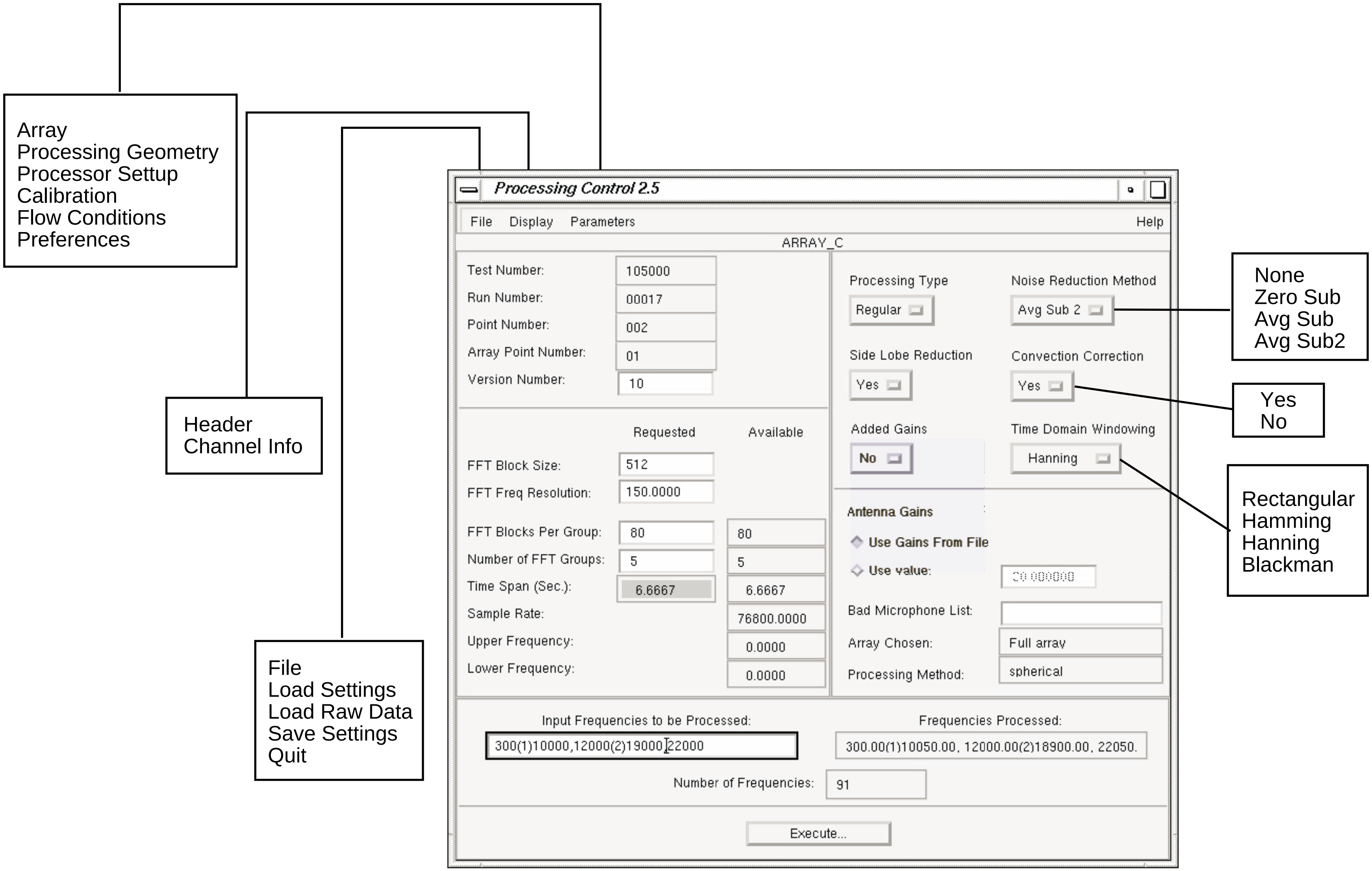


Figure 7. Block decimation scheme.



ARRAY_C

Test Number: 105000
Run Number: 00017
Point Number: 002
Array Point Number: 01
Version Number: 10

	Requested	Available
FFT Block Size:	512	
FFT Freq Resolution:	150.0000	
FFT Blocks Per Group:	80	80
Number of FFT Groups:	5	5
Time Span (Sec.):	6.6667	6.6667
Sample Rate:		76800.0000
Upper Frequency:		0.0000
Lower Frequency:		0.0000

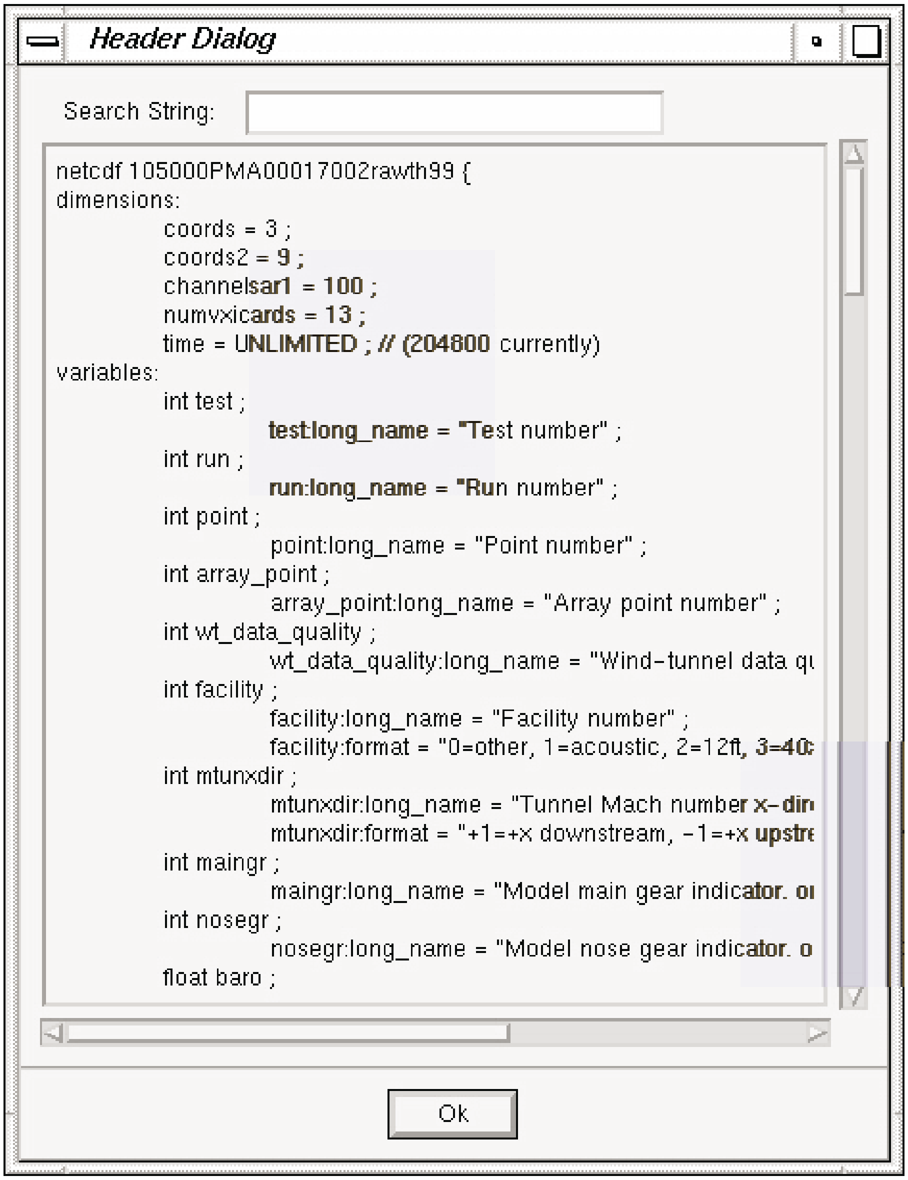
Processing Type: Regular
Noise Reduction Method: Avg Sub 2
Side Lobe Reduction: Yes
Convection Correction: Yes
Added Gains: No
Time Domain Windowing: Hanning

Antenna Gains
◆ Use Gains From File
◇ Use value: 20.000000

Bad Microphone List:
Array Chosen: Full array
Processing Method: spherical

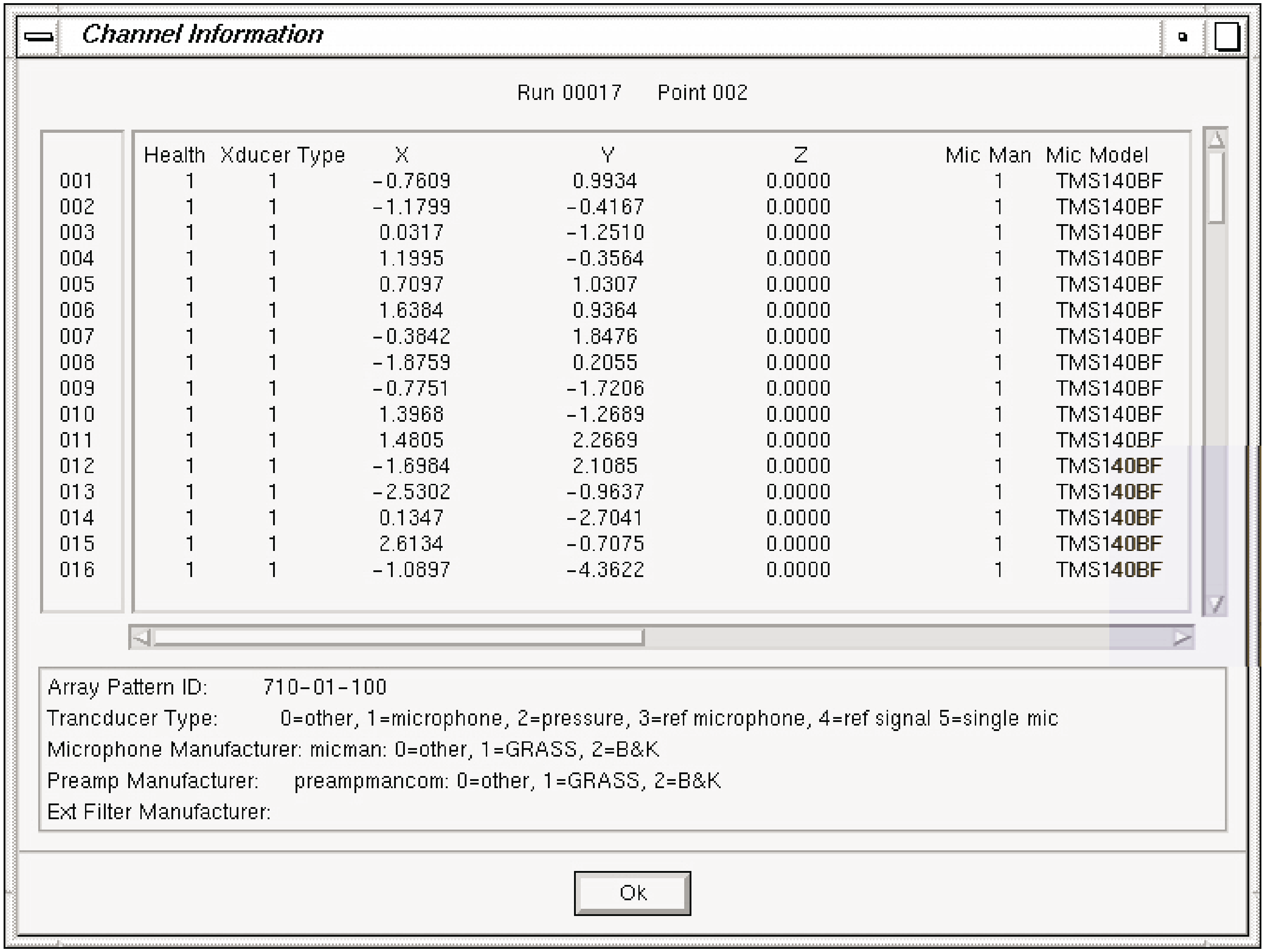
Input Frequencies to be Processed: 300(1)10000,12000(2)19000,22000
Frequencies Processed: 300.00(1)10050.00, 12000.00(2)18900.00, 22050.
Number of Frequencies: 91

Figure 8. Control interface main window.



```
netcdf 105000PMA00017002rawth99 {
dimensions:
        coords = 3 ;
        coords2 = 9 ;
        channelsar1 = 100 ;
        numvxicards = 13 ;
        time = UNLIMITED ; // (204800 currently)
variables:
        int test ;
                test:long_name = "Test number" ;
        int run ;
                run:long_name = "Run number" ;
        int point ;
                point:long_name = "Point number" ;
        int array_point ;
                array_point:long_name = "Array point number" ;
        int wt_data_quality ;
                wt_data_quality:long_name = "Wind-tunnel data qu
        int facility ;
                facility:long_name = "Facility number" ;
                facility:format = "0=other, 1=acoustic, 2=12ft, 3=40
        int mtunxdir ;
                mtunxdir:long_name = "Tunnel Mach number x-dir
                mtunxdir:format = "+1=+x downstream, -1=+x upstre
        int maingr ;
                maingr:long_name = "Model main gear indicator. o
        int nosegr ;
                nosegr:long_name = "Model nose gear indicator. o
        float baro ;
```

Figure 9. Control interface header information window.

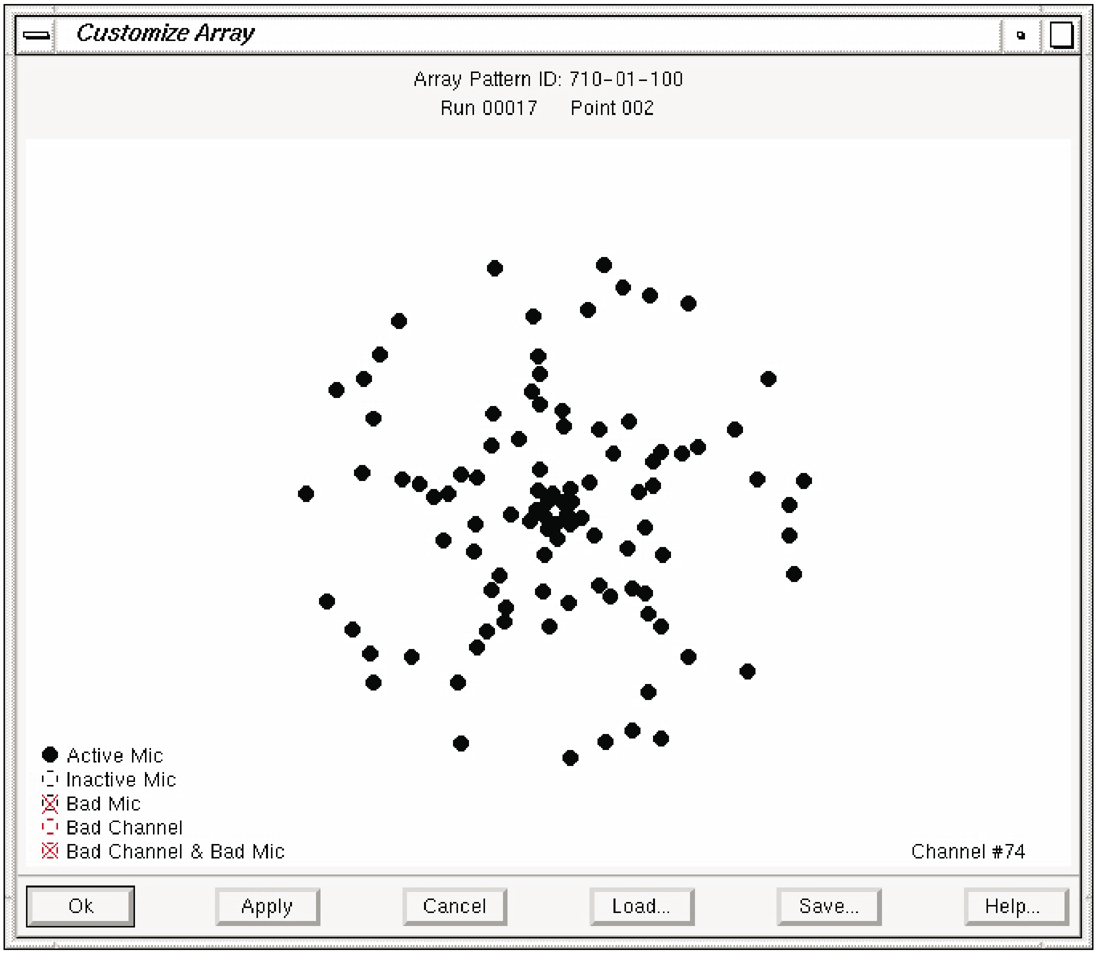


Run 00017   Point 002

	Health	Xducer Type	X	Y	Z	Mic Man	Mic Model
001	1	1	-0.7609	0.9934	0.0000	1	TMS140BF
002	1	1	-1.1799	-0.4167	0.0000	1	TMS140BF
003	1	1	0.0317	-1.2510	0.0000	1	TMS140BF
004	1	1	1.1995	-0.3564	0.0000	1	TMS140BF
005	1	1	0.7097	1.0307	0.0000	1	TMS140BF
006	1	1	1.6384	0.9364	0.0000	1	TMS140BF
007	1	1	-0.3842	1.8476	0.0000	1	TMS140BF
008	1	1	-1.8759	0.2055	0.0000	1	TMS140BF
009	1	1	-0.7751	-1.7206	0.0000	1	TMS140BF
010	1	1	1.3968	-1.2689	0.0000	1	TMS140BF
011	1	1	1.4805	2.2669	0.0000	1	TMS140BF
012	1	1	-1.6984	2.1085	0.0000	1	TMS140BF
013	1	1	-2.5302	-0.9637	0.0000	1	TMS140BF
014	1	1	0.1347	-2.7041	0.0000	1	TMS140BF
015	1	1	2.6134	-0.7075	0.0000	1	TMS140BF
016	1	1	-1.0897	-4.3622	0.0000	1	TMS140BF

Array Pattern ID: 710-01-100
Trancducer Type: 0=other, 1=microphone, 2=pressure, 3=ref microphone, 4=ref signal 5=single mic
Microphone Manufacturer: micman: 0=other, 1=GRASS, 2=B&K
Preamp Manufacturer: preampmancom: 0=other, 1=GRASS, 2=B&K
Ext Filter Manufacturer:

Figure 10. Control interface instrumentation information window.



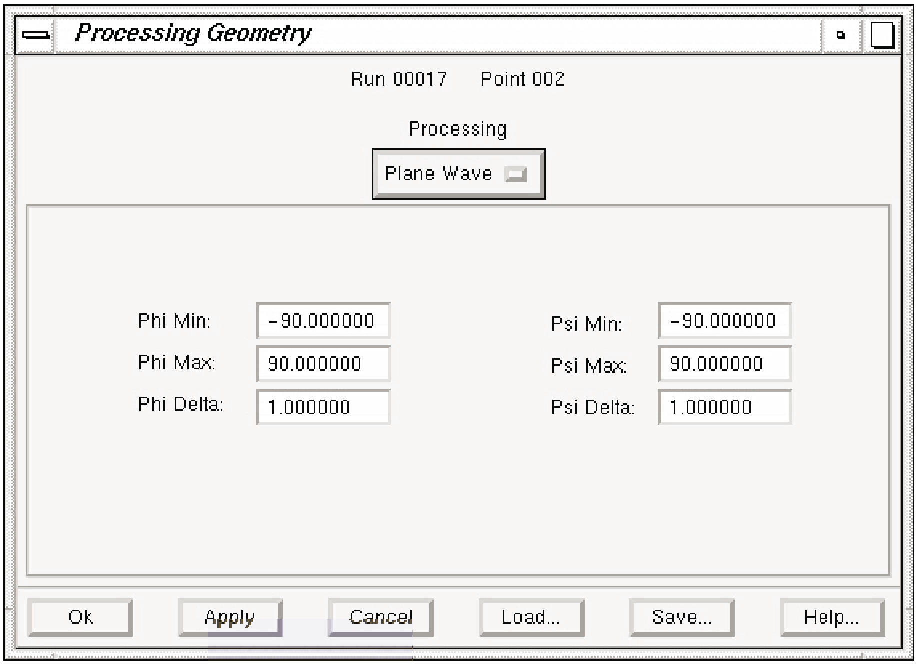
Customize Array

Array Pattern ID: 710-01-100
Run 00017 Point 002

● Active Mic
○ Inactive Mic
☒ Bad Mic
○ Bad Channel
☒ Bad Channel & Bad Mic

Channel #74

Figure 11. Control interface customize array window.



Processing Geometry

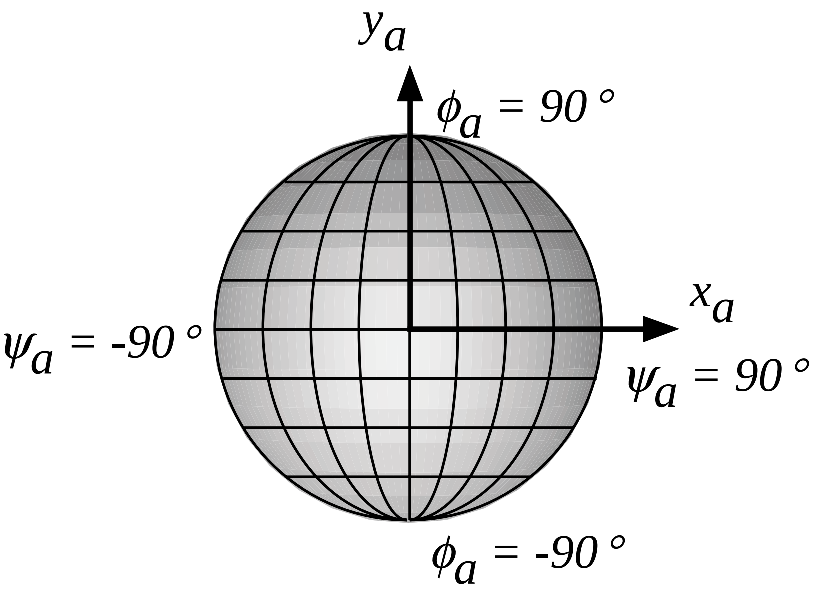
Run 00017 Point 002

Processing

Plane Wave

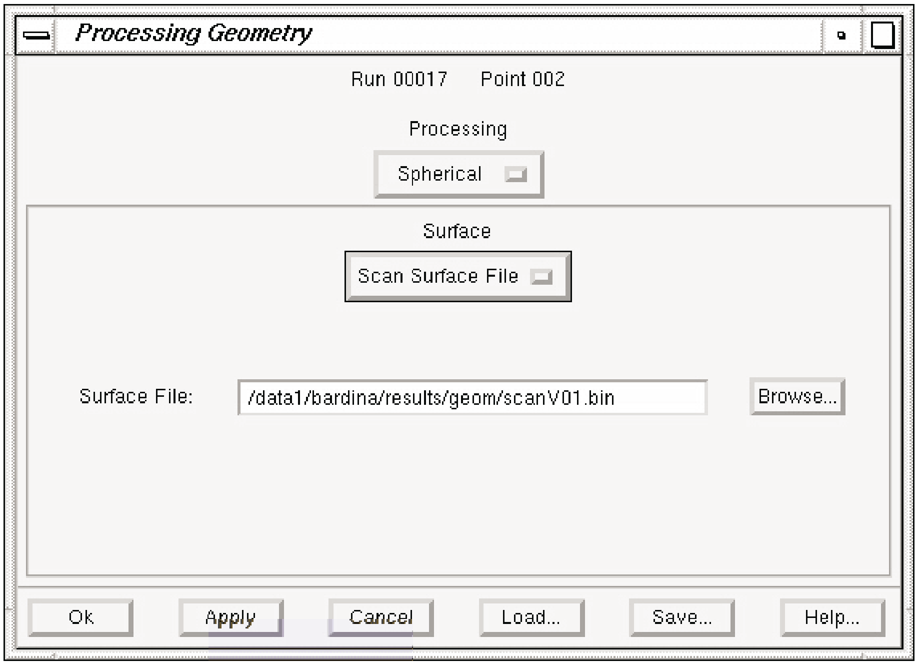
Phi Min:	-90.000000	Psi Min:	-90.000000
Phi Max:	90.000000	Psi Max:	90.000000
Phi Delta:	1.000000	Psi Delta:	1.000000

a) Control interface plane wave scan definition window.



b) Plane wave angle definition.

Figure 12. Plane wave geometry definition.



Processing Geometry

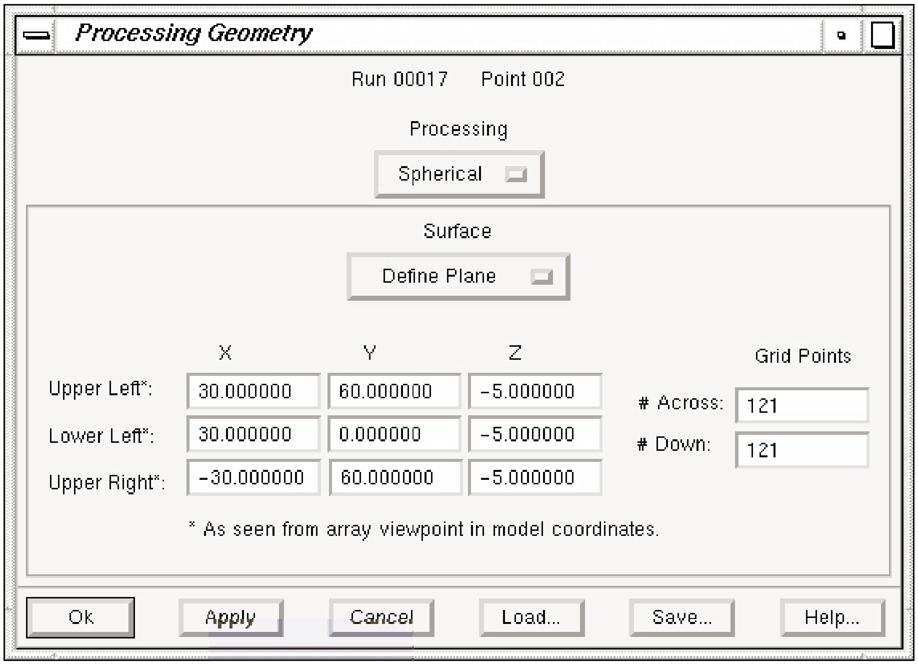
Run 00017 Point 002

Processing: Spherical

Surface: Scan Surface File

Surface File: /data1/bardina/results/geom/scanV01.bin

Figure 13. Control interface scan surface file input window.



Processing Geometry

Run 00017 Point 002

Processing: Spherical

Surface: Define Plane

	X	Y	Z
Upper Left*:	30.000000	60.000000	-5.000000
Lower Left*:	30.000000	0.000000	-5.000000
Upper Right*:	-30.000000	60.000000	-5.000000

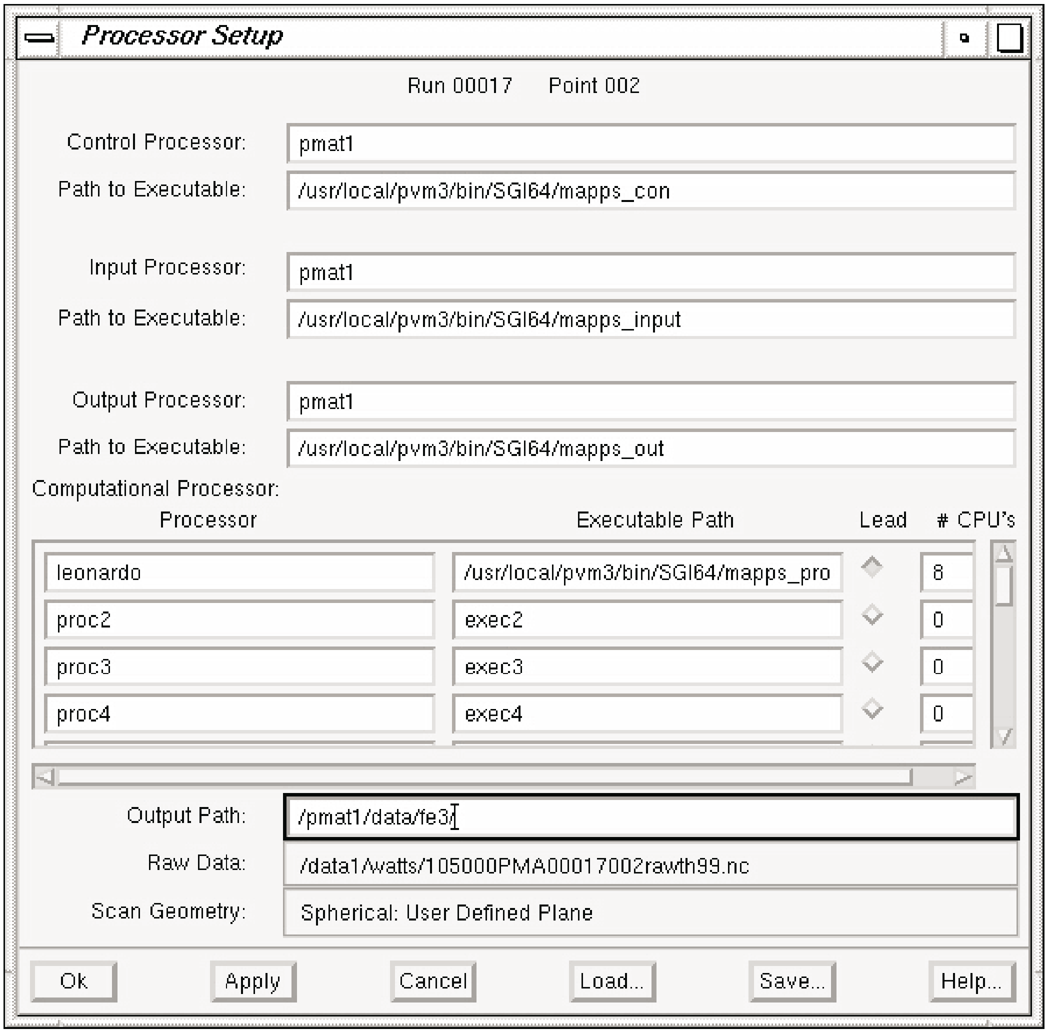
Grid Points

# Across: 121

# Down: 121

* As seen from array viewpoint in model coordinates.

Figure 14. Control interface scan surface corner input window.



Run 00017 Point 002

Control Processor: pmat1

Path to Executable: /usr/local/pvm3/bin/SGI64/mapps_con

Input Processor: pmat1

Path to Executable: /usr/local/pvm3/bin/SGI64/mapps_input

Output Processor: pmat1

Path to Executable: /usr/local/pvm3/bin/SGI64/mapps_out

Computational Processor:

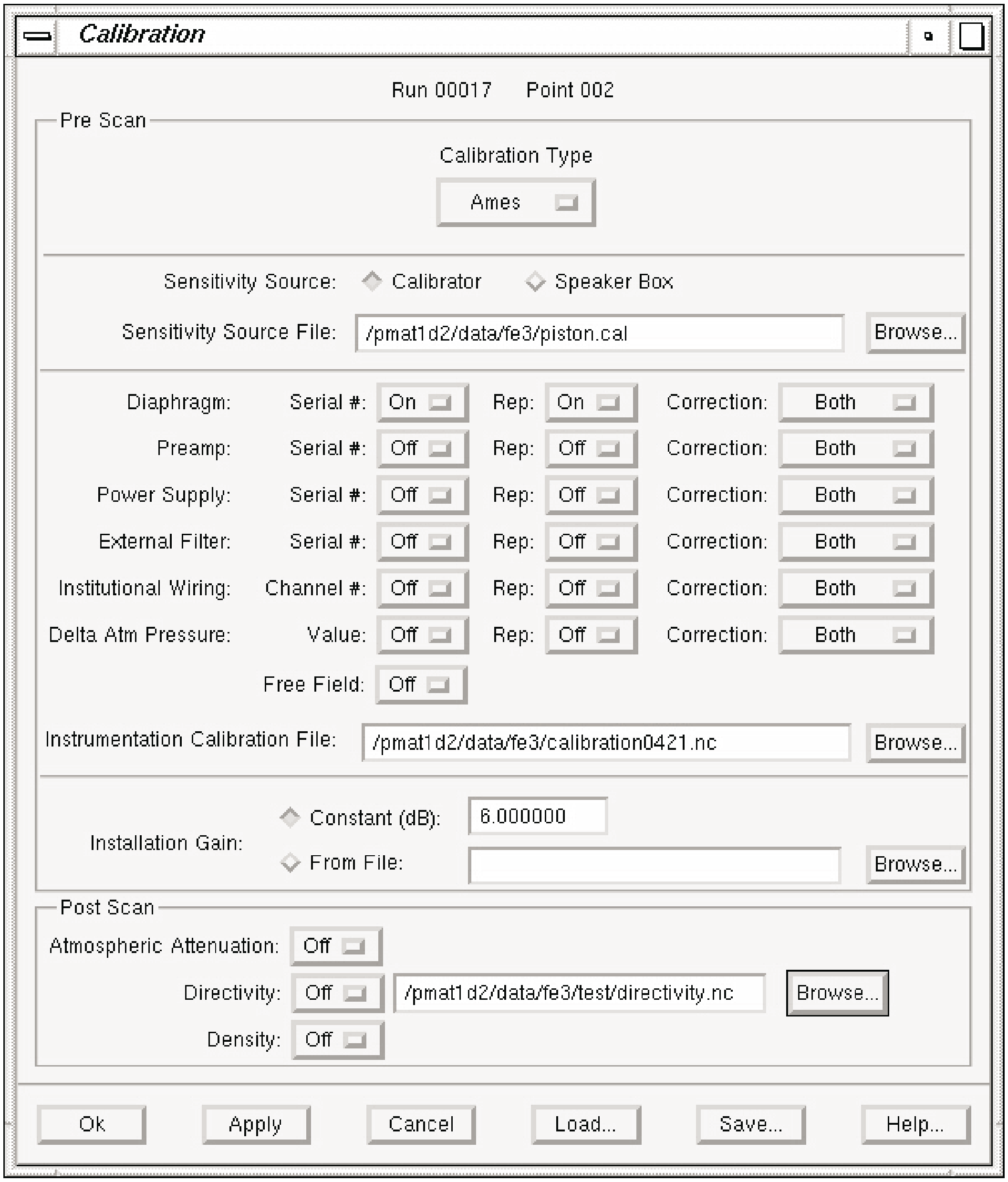
Processor	Executable Path	Lead	# CPU's
leonardo	/usr/local/pvm3/bin/SGI64/mapps_pro	◆	8
proc2	exec2	◇	0
proc3	exec3	◇	0
proc4	exec4	◇	0

Output Path: /pmat1/data/fe3/

Raw Data: /data1/watts/105000PMA00017002rawth99.nc

Scan Geometry: Spherical: User Defined Plane

Figure 15. Control interface processor setup window.



Run 00017 Point 002

Pre Scan

Calibration Type: Ames

Sensitivity Source: ◆ Calibrator ◇ Speaker Box

Sensitivity Source File: /pmat1d2/data/fe3/piston.cal

Diaphragm:	Serial #:	On	Rep:	On	Correction:	Both
Preamp:	Serial #:	Off	Rep:	Off	Correction:	Both
Power Supply:	Serial #:	Off	Rep:	Off	Correction:	Both
External Filter:	Serial #:	Off	Rep:	Off	Correction:	Both
Institutional Wiring:	Channel #:	Off	Rep:	Off	Correction:	Both
Delta Atm Pressure:	Value:	Off	Rep:	Off	Correction:	Both
	Free Field:	Off				

Instrumentation Calibration File: /pmat1d2/data/fe3/calibration0421.nc

Installation Gain: ◆ Constant (dB): 6.000000
◇ From File:

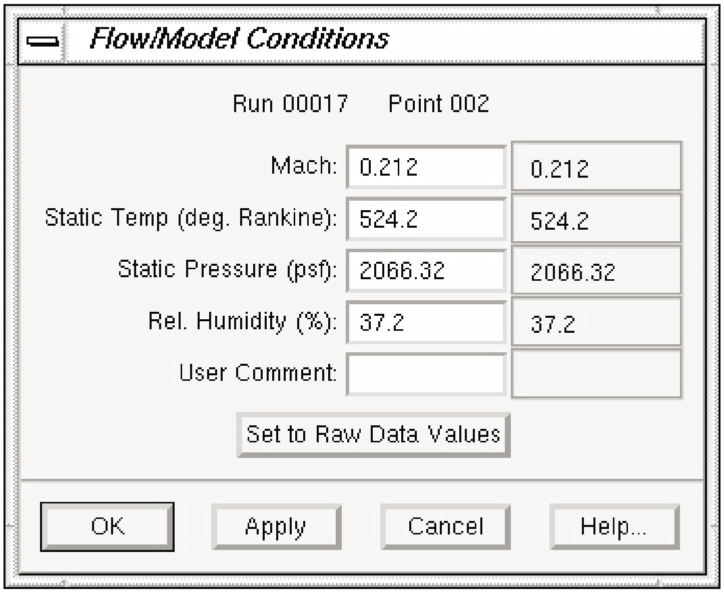
Post Scan

Atmospheric Attenuation: Off

Directivity: Off /pmat1d2/data/fe3/test/directivity.nc

Density: Off

Figure 16. Control interface calibration window.

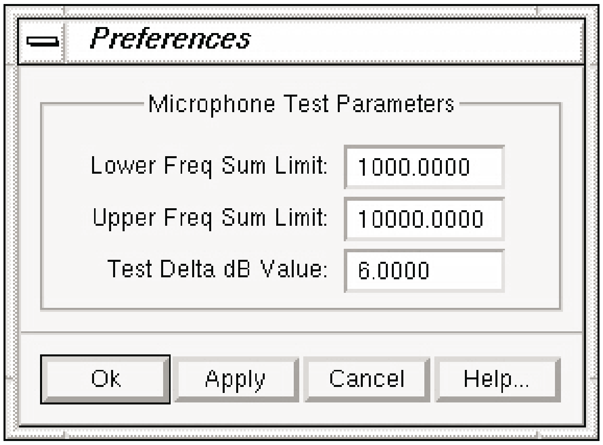


Flow/Model Conditions

Run 00017 Point 002

Mach:	0.212	0.212
Static Temp (deg. Rankine):	524.2	524.2
Static Pressure (psf):	2066.32	2066.32
Rel. Humidity (%):	37.2	37.2
User Comment:		

Figure 17. Control interface flow conditions window.

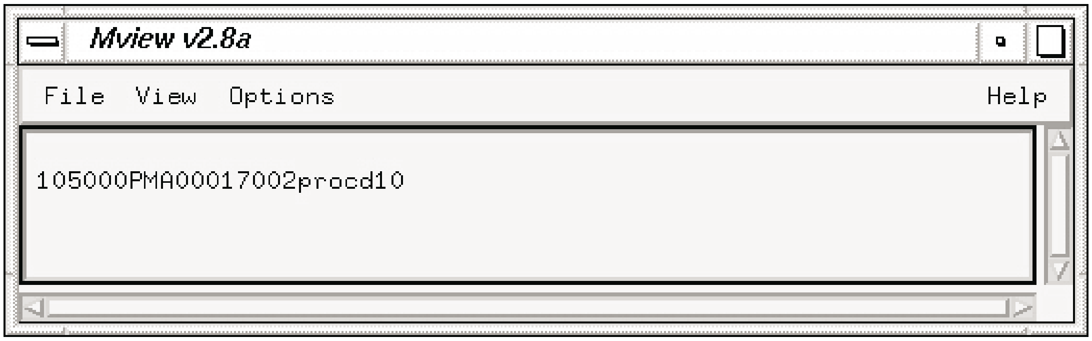


Preferences

Microphone Test Parameters

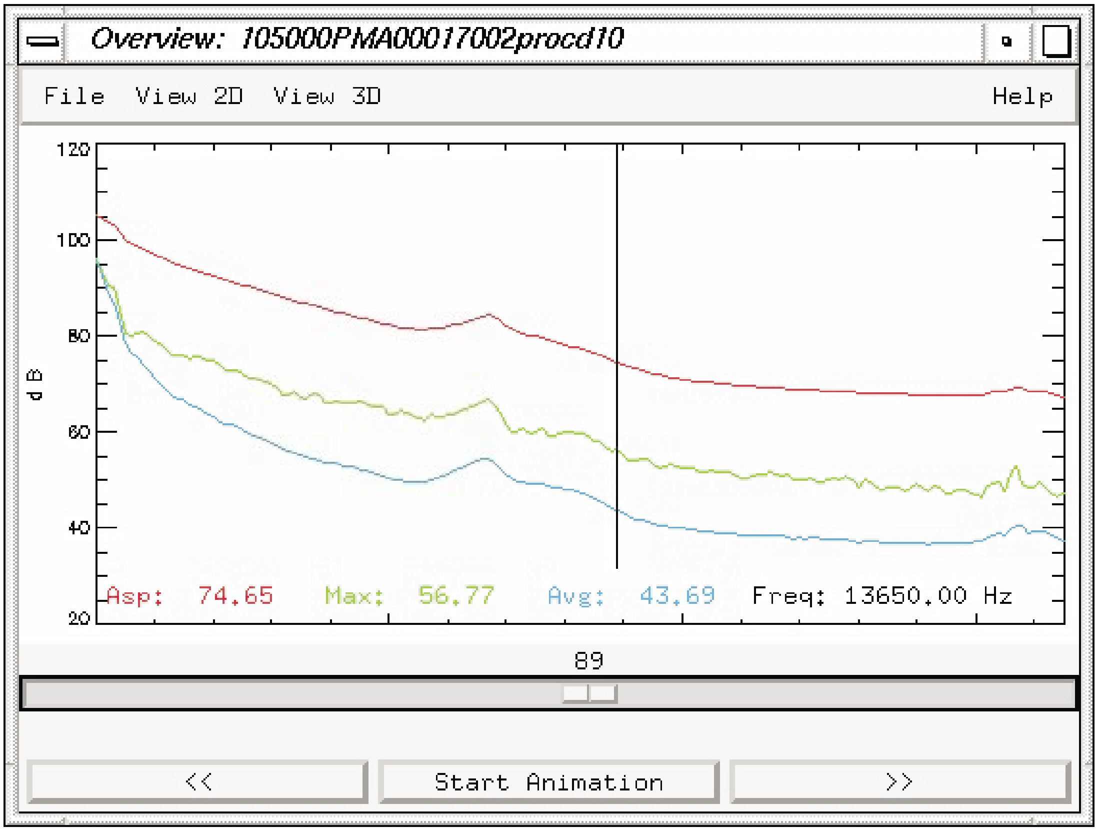
Lower Freq Sum Limit:	1000.0000
Upper Freq Sum Limit:	10000.0000
Test Delta dB Value:	6.0000

Figure 18. Control interface preferences window.



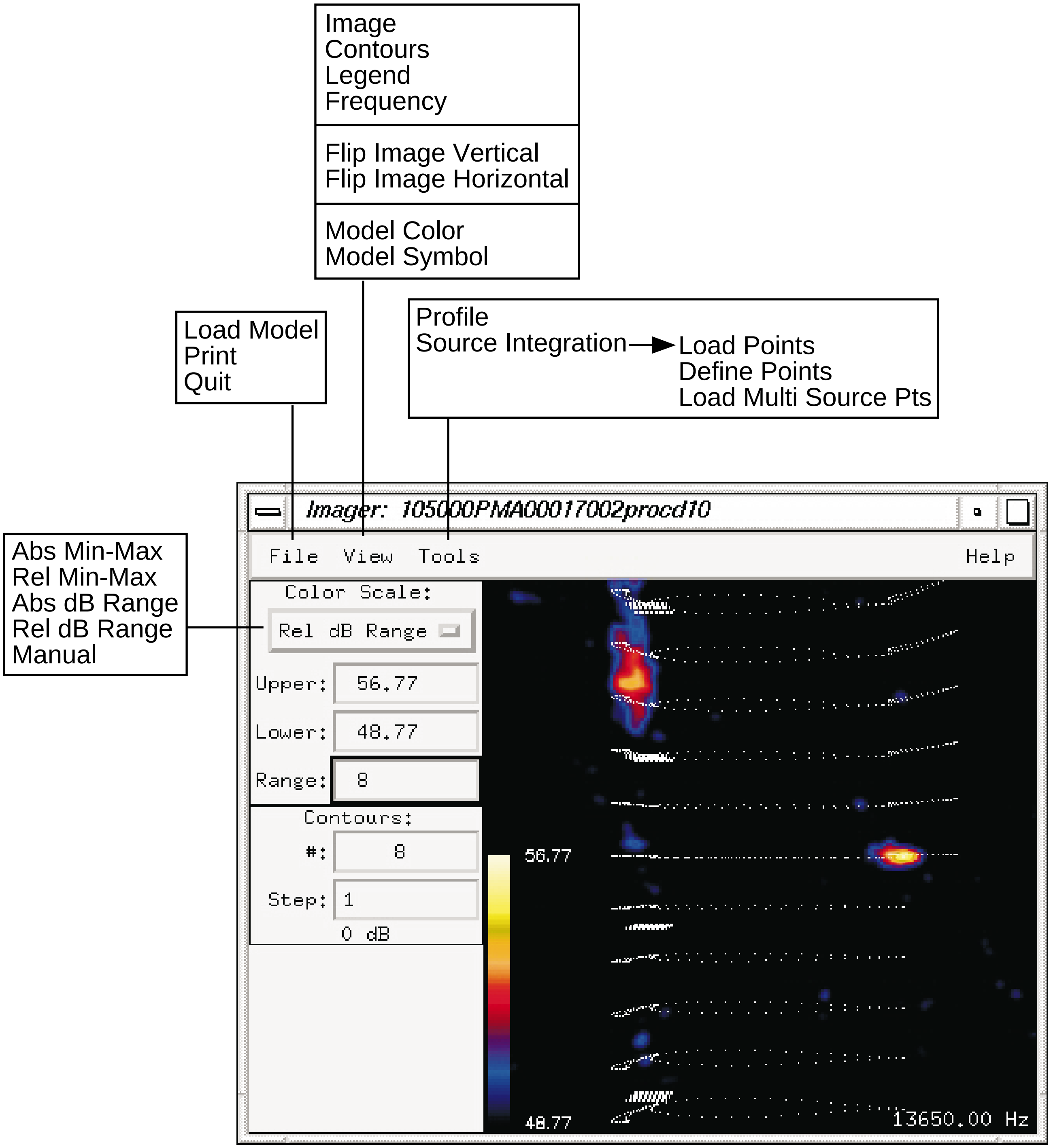
105000PMA00017002procd10

Figure 19. Mview main window.



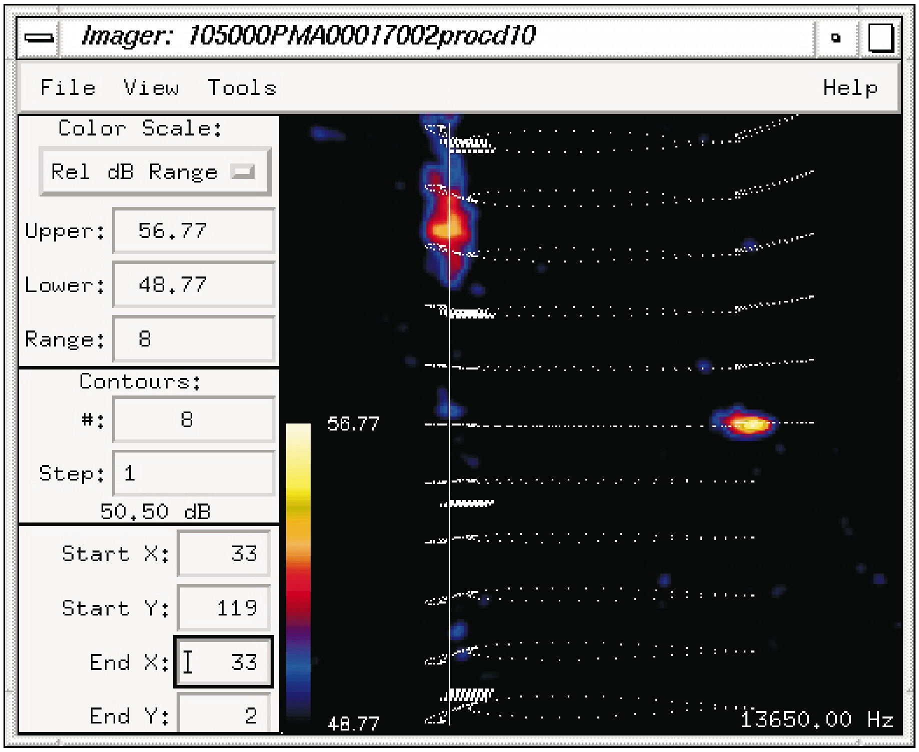
Asp: 74.65 Max: 56.77 Avg: 43.69 Freq: 13650.00 Hz

Figure 20. Mview overview window.



Color Scale: Rel dB Range
Upper: 56.77
Lower: 48.77
Range: 8
Contours:
#: 8
Step: 1
0 dB

Figure 21. Mview imager window.



Imager: 105000PMA00017002procd10

Color Scale: Rel dB Range

Upper: 56.77

Lower: 48.77

Range: 8

Contours:

#: 8

Step: 1

50.50 dB

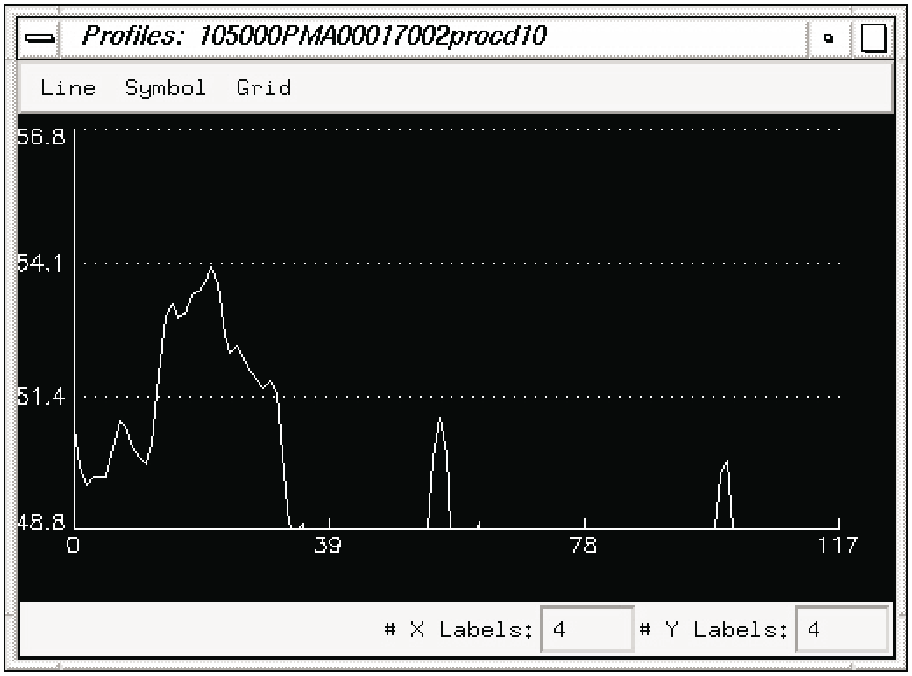
Start X: 33

Start Y: 119

End X: 33

End Y: 2

Figure 22. Mview imager window with profiles activated.

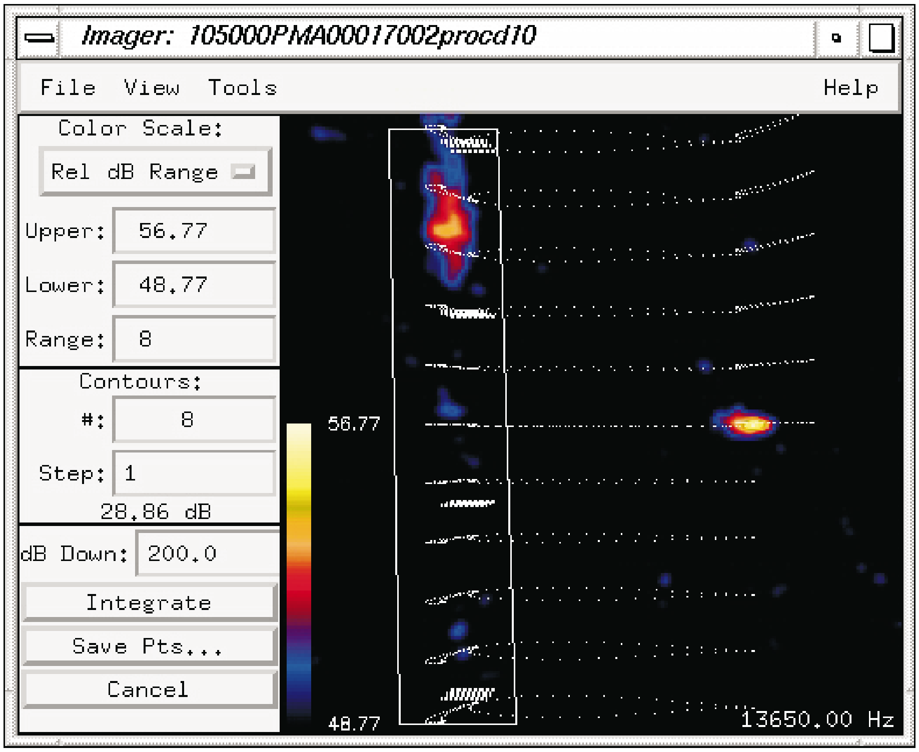


Profiles: 105000PMA00017002procd10

X Labels: 4

Y Labels: 4

Figure 23. Mview profiles window.



Color Scale: Rel dB Range

Upper: 56.77

Lower: 48.77

Range: 8

Contours:

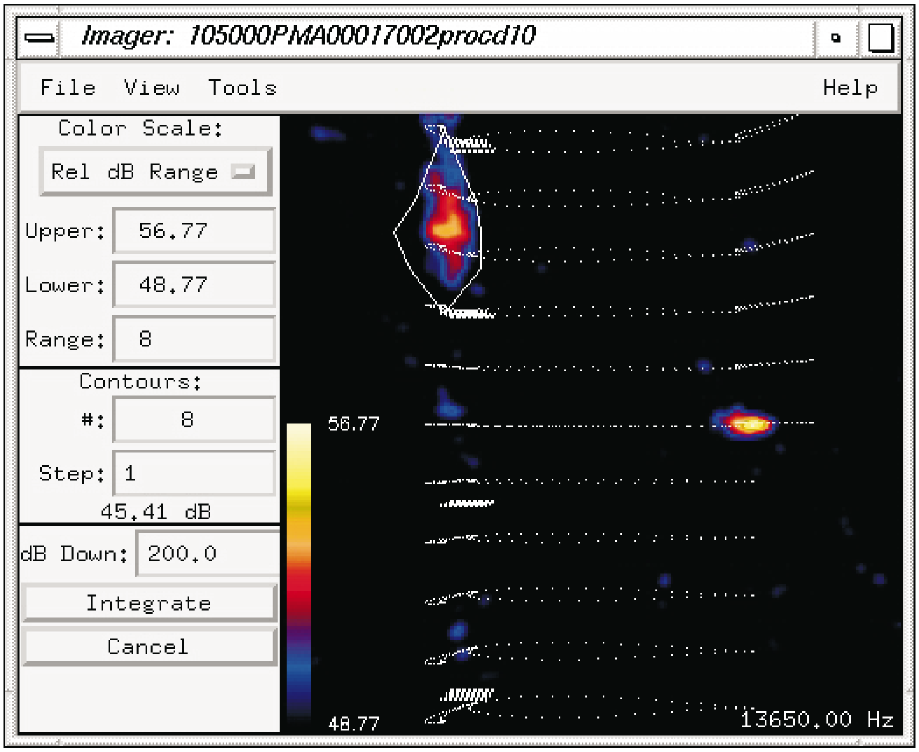
#: 8

Step: 1

28.86 dB

dB Down: 200.0

Figure 24. Mview imager window in source integration define points mode.



Color Scale: Rel dB Range

Upper: 56.77

Lower: 48.77

Range: 8

Contours:

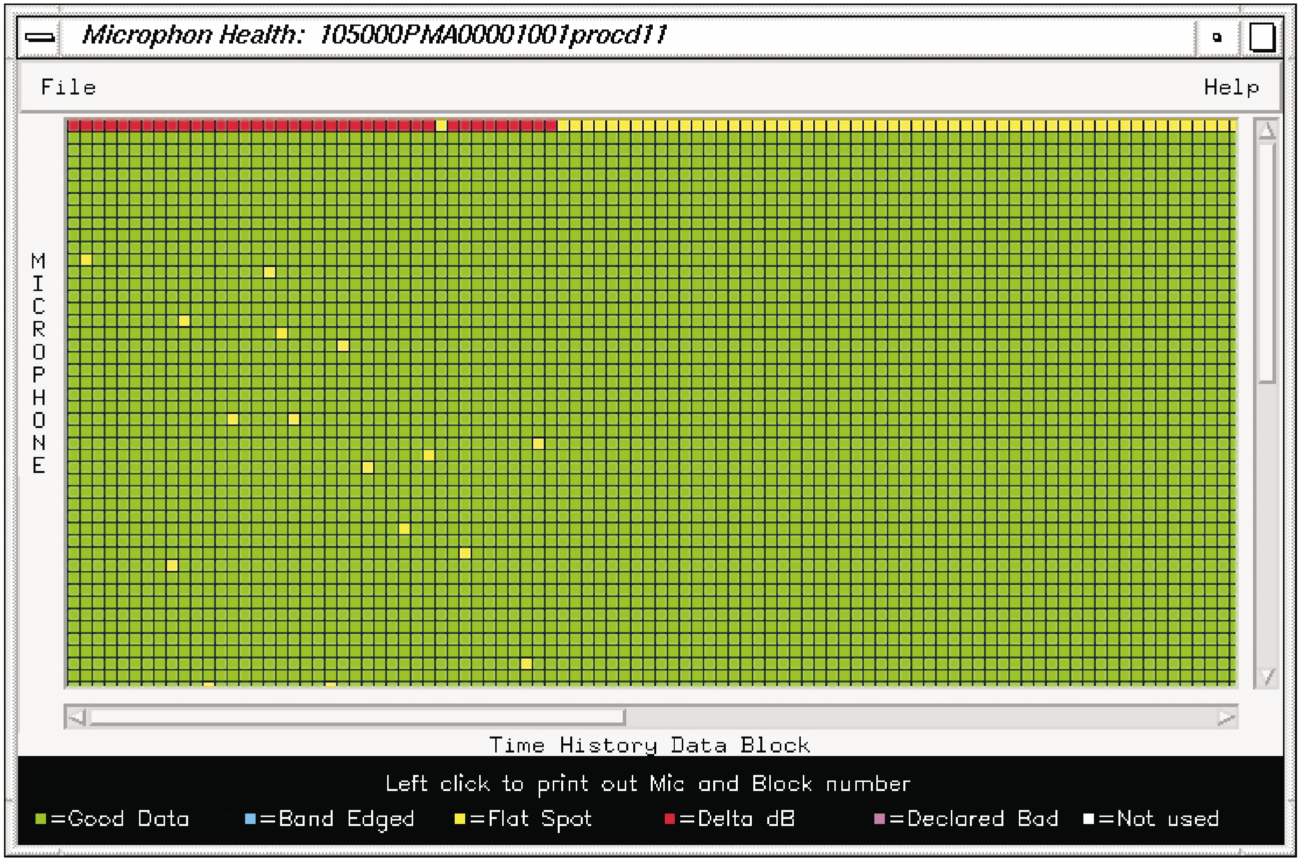
#: 8

Step: 1

45.41 dB

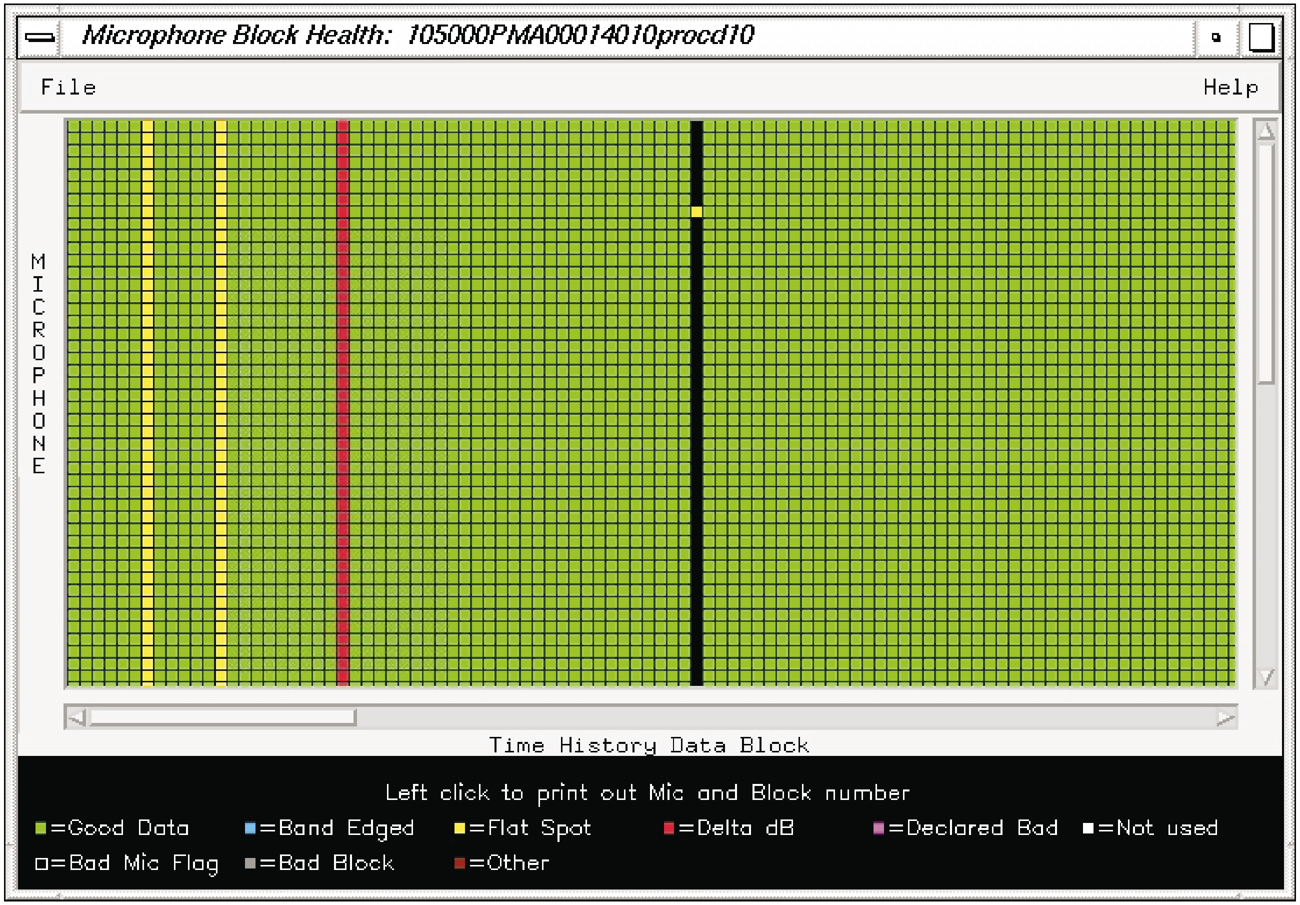
dB Down: 200.0

Figure 25. Mview imager window in source integration multisource mode.



Left click to print out Mic and Block number

Figure 26. Mview individual microphone health window.

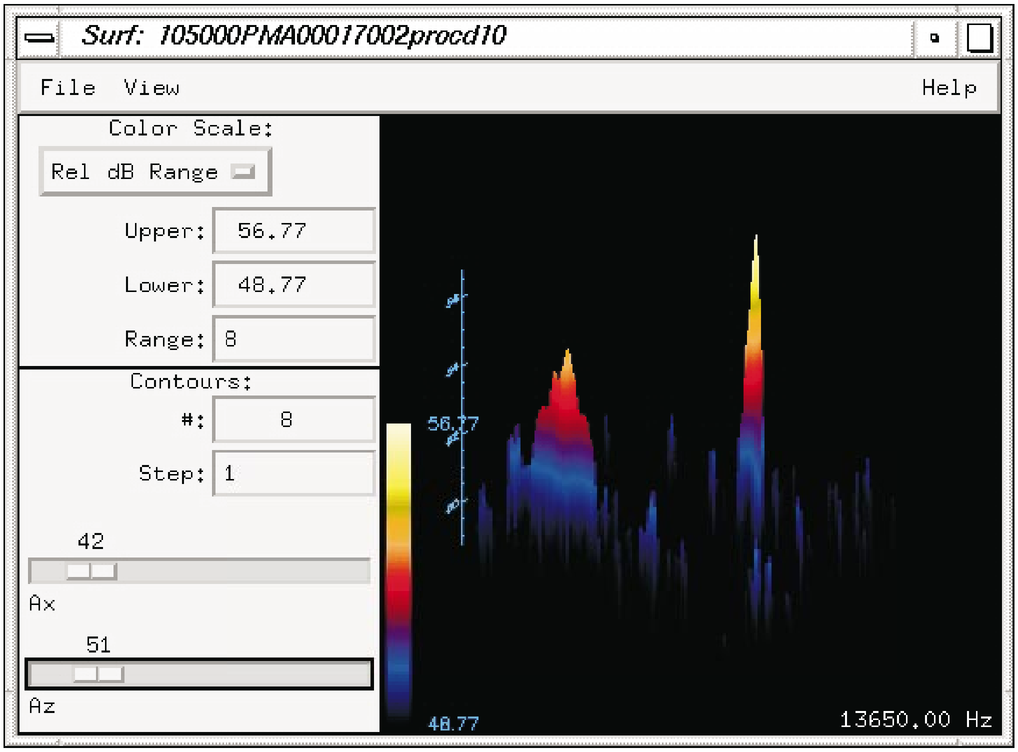


Left click to print out Mic and Block number

🟩=Good Data 🟦=Band Edged 🟨=Flat Spot 🟥=Delta dB 🟪=Declared Bad ■=Not used

□=Bad Mic Flag ▨=Bad Block 🟫=Other

Figure 27. Mview combined microphone health window.



Surf: 105000PMA00017002procd10

Color Scale: Rel dB Range

Upper: 56.77

Lower: 48.77

Range: 8

Contours:

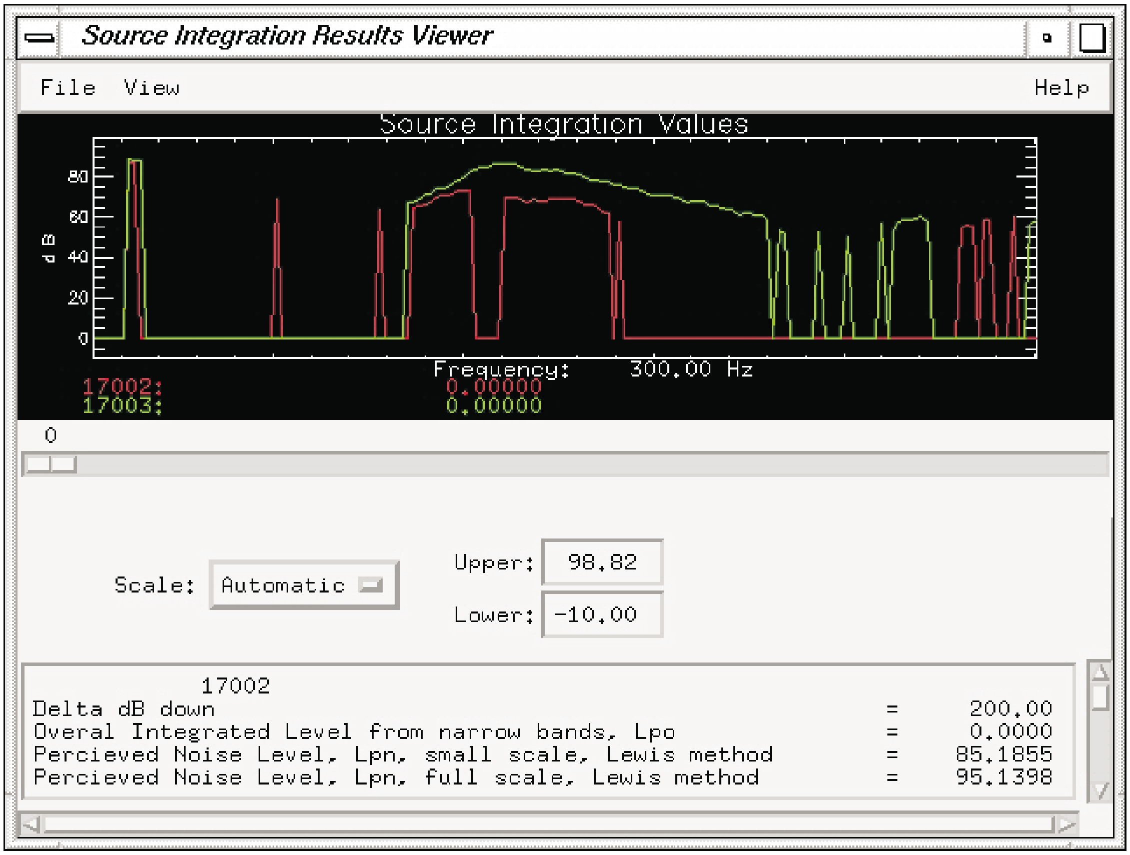
#: 8

Step: 1

Ax: 42

Az: 51

Figure 28. Mview surface plot window.



```
Frequency:     300.00 Hz
17002:     0.00000
17003:     0.00000
```

```
0
```

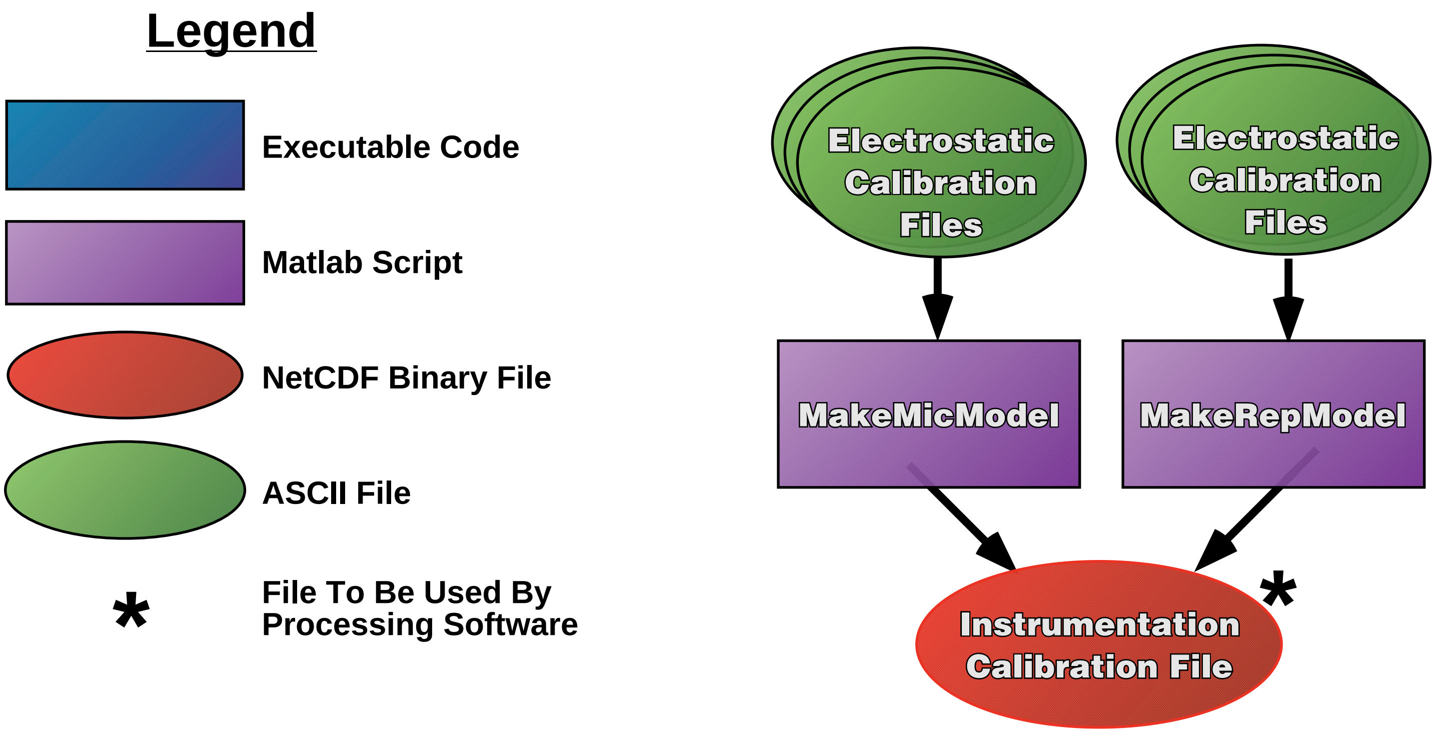
Scale: Automatic

Upper: 98.82

Lower: -10.00

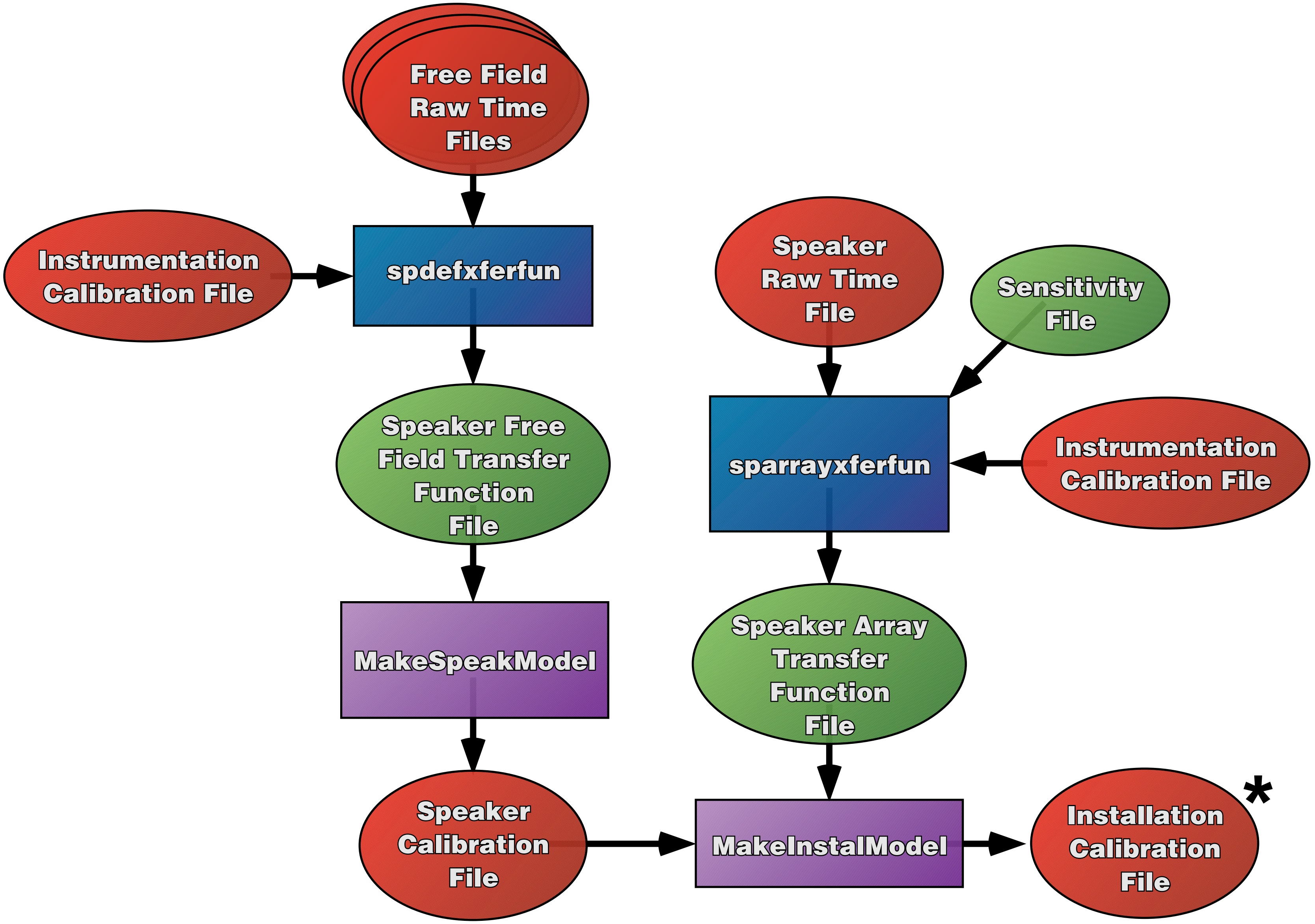
```
             17002
Delta dB down                                                      =     200.00
Overal Integrated Level from narrow bands, Lpo                     =     0.0000
Percieved Noise Level, Lpn, small scale, Lewis method              =    85.1855
Percieved Noise Level, Lpn, full scale, Lewis method               =    95.1398
```

Figure 29. Mview source integration results window.


Legend
Executable Code
Matlab Script
NetCDF Binary File
ASCII File
*
File To Be Used By Processing Software
Electrostatic Calibration Files
Electrostatic Calibration Files
MakeMicModel
MakeRepModel
Instrumentation Calibration File
*

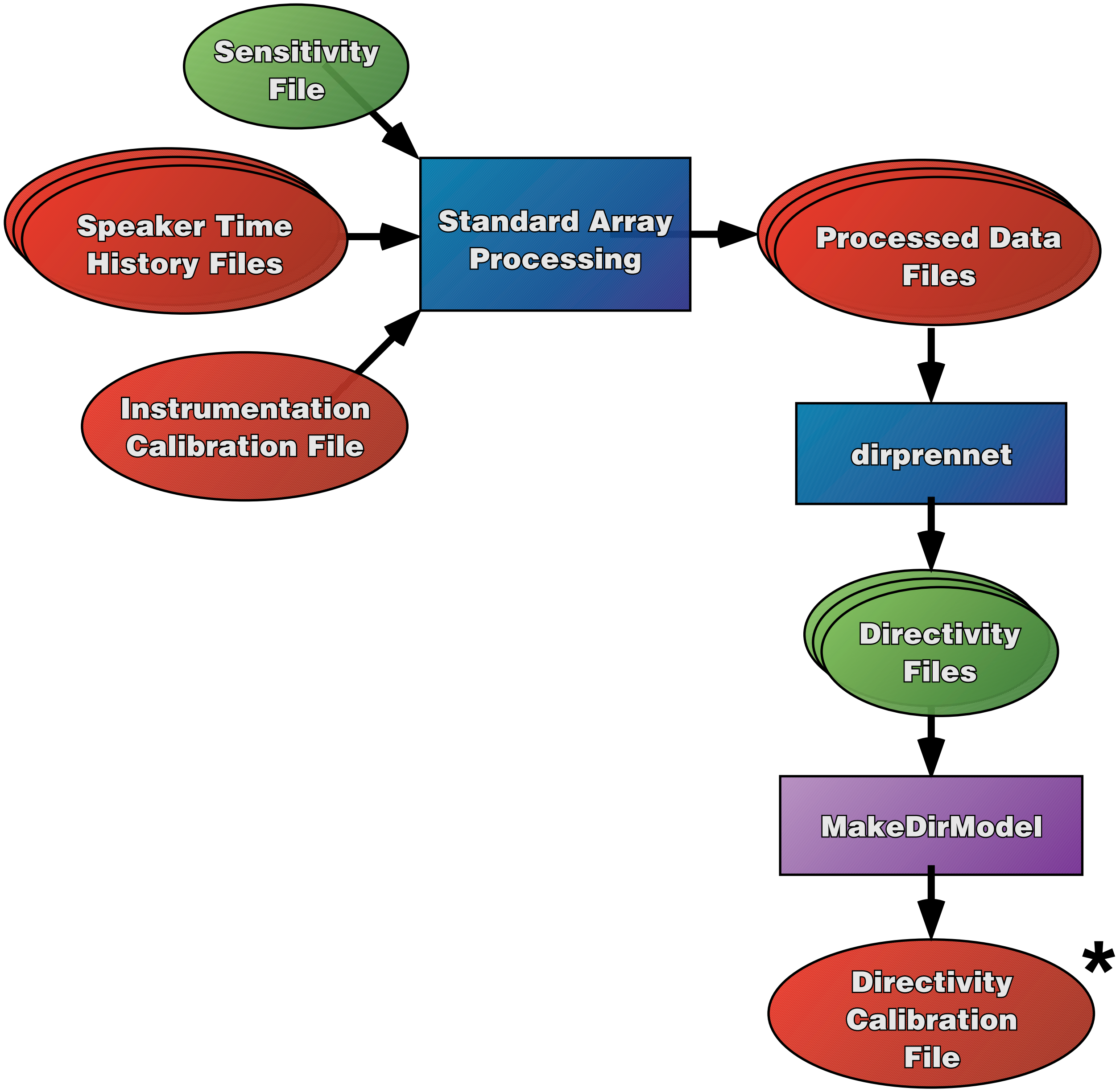
a) Individual instrumentation correction.

Figure 30. Calibration file generation flow chart.


Free Field Raw Time Files
Instrumentation Calibration File
spdefxferfun
Speaker Raw Time File
Sensitivity File
Speaker Free Field Transfer Function File
sparrayxferfun
Instrumentation Calibration File
MakeSpeakModel
Speaker Array Transfer Function File
Speaker Calibration File
MakeInstalModel
Installation Calibration File
*

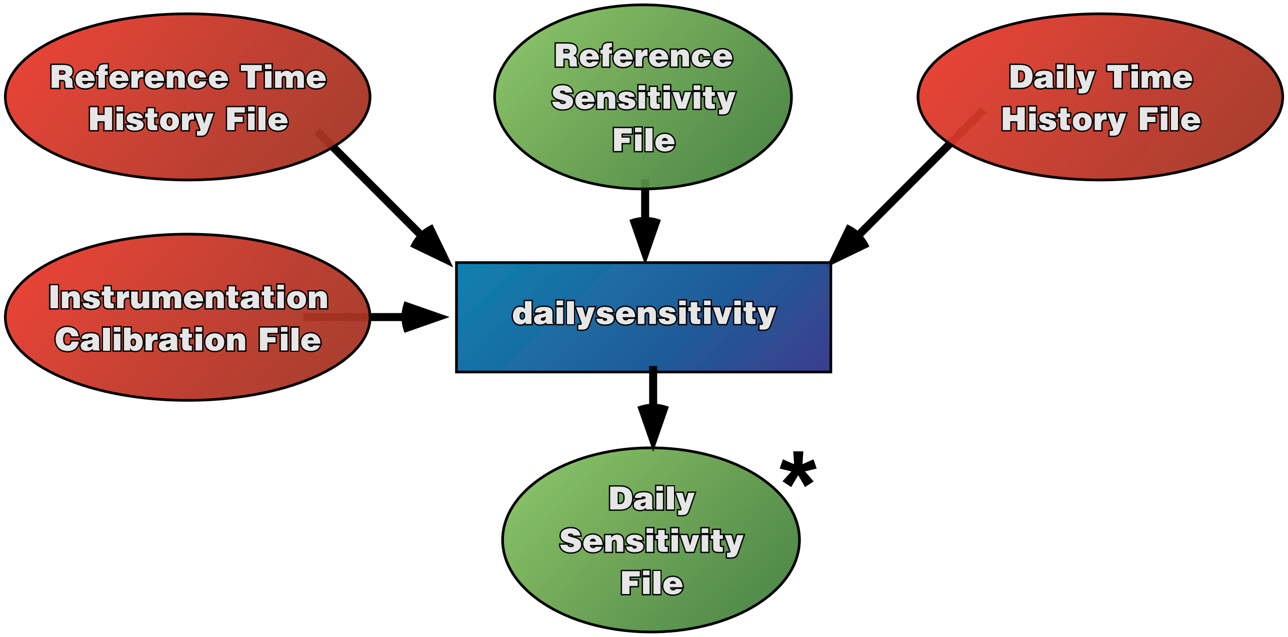
b) Installation correction.

Figure 30. Calibration file generation flow chart (continued).


Sensitivity File
Speaker Time History Files
Instrumentation Calibration File
Standard Array Processing
Processed Data Files
dirprennet
Directivity Files
MakeDirModel
Directivity Calibration File
*

c) Directivity Correction.

Figure 30. Calibration file generation flow chart (continued).


Reference Time History File
Reference Sensitivity File
Daily Time History File
Instrumentation Calibration File
dailysensitivity
Daily Sensitivity File
*

d) Sensitivity Correction.

Figure 30. Calibration file generation flow chart (concluded).

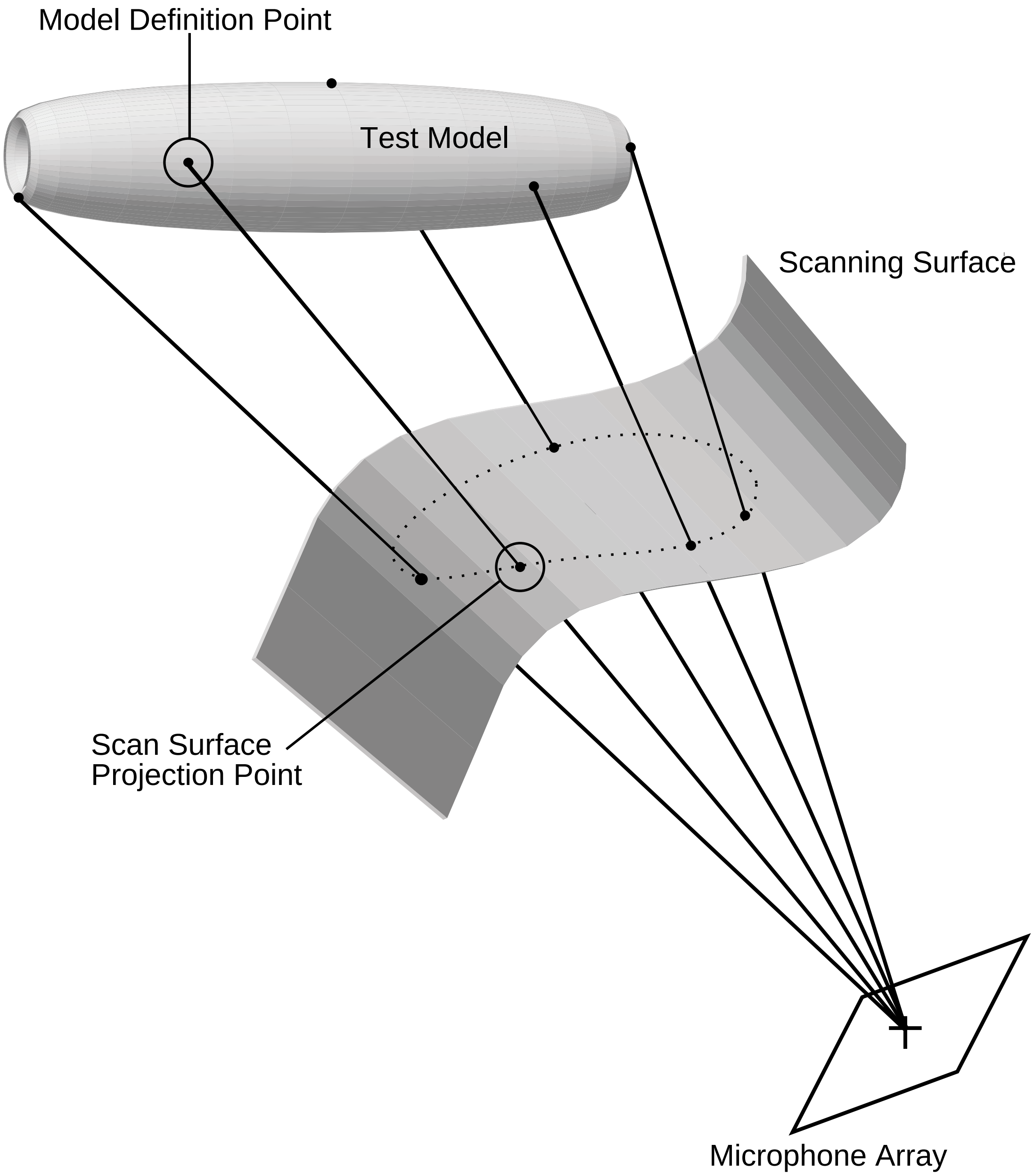


Figure 31. Scan surface projected points scheme.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1999	3. REPORT TYPE AND DATES COVERED Technical Memorandum

4. TITLE AND SUBTITLE

Microphone Array Phased Processing System (MAPPS)
Version 4.0 Manual

5. FUNDING NUMBERS

519-20-21

6. AUTHOR(S)

Michael E. Watts, Marianne Mosher, Michael Barnes,*
and Jorge Bardina*

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Ames Research Center
Moffett Field, CA 94035-1000

8. PERFORMING ORGANIZATION REPORT NUMBER

A-9900429

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA/TM–1999-208755

11. SUPPLEMENTARY NOTES

Point of Contact: Michael E. Watts, Ames Research Center, MS 269-3, Moffett Field, CA 94035-1000
(650) 604-6574

*Caelum Research Corporation, Ames Research Center

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified — Unlimited
Subject Category 02 Distribution: Standard
Availability: NASA CASI (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A processing system has been developed to meet increasing demands for detailed noise measurement of individual model components. The Microphone Array Phased Processing System (MAPPS) uses graphical user interfaces to control all aspects of data processing and visualization. The system uses networked parallel computers to provide noise maps at selected frequencies in a near real-time testing environment. The system has been successfully used in the NASA Ames 7- by 10-Foot Wind Tunnel.

14. SUBJECT TERMS

Microphone array, Acoustics, Parallel processing

15. NUMBER OF PAGES

131

16. PRICE CODE

A07

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT