



3 4456 0428220 1

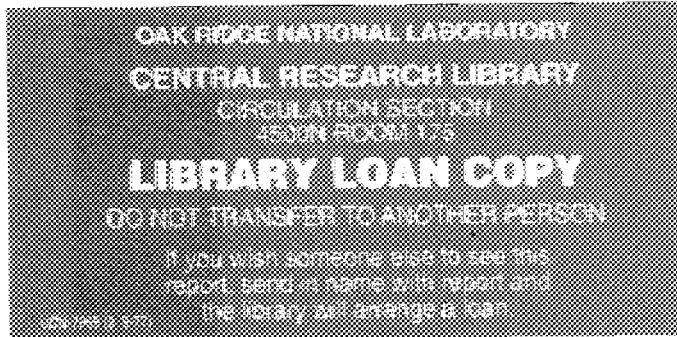
ORNL/TM-13572

**oml1****OAK RIDGE  
NATIONAL  
LABORATORY**

LOCKHEED MARTIN

**PCB DATA INTERPRETATION  
STANDARD LABORATORY MODULE:  
CONTROL AND COMMUNICATIONS**

Martin A. Hunt  
Oak Ridge National Laboratory



MANAGED AND OPERATED BY  
LOCKHEED MARTIN ENERGY RESEARCH CORPORATION  
FOR THE UNITED STATES  
DEPARTMENT OF ENERGY

Prepared by Oak Ridge National Laboratory,  
Oak Ridge, Tennessee 37831-6285, managed by  
Lockheed Martin Energy Research Corp. for the  
U.S. Department of Energy under contract DE-  
AC05-96OR22464.

This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831; prices available from (615) 576-8401, FTS 626-8401.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161.

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

**PCB DATA INTERPRETATION STANDARD LABORATORY MODULE:  
CONTROL AND COMMUNICATIONS**

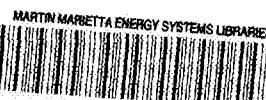
Martin A. Hunt  
Oak Ridge National Laboratory

January, 1998

"The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. DE-AC05-96OR22464. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes."

---

Prepared by OAK RIDGE NATIONAL LABORATORY, Oak Ridge, Tennessee 37831-6285, managed by LOCKHEED MARTIN ENERGY RESEARCH CORP. for the U.S. DEPARTMENT OF ENERGY under contract DE-AC05-96OR22464.



3 4456 0428220 1



## Table of Contents

<b>1. INTRODUCTION.....</b>	<b>2</b>
<b>2. FUNCTIONAL DESCRIPTION .....</b>	<b>2</b>
2.1 TSC-DIM INTERFACE .....	2
2.2 DIM-MATLAB INTERFACE .....	3
2.3 DATA MANAGEMENT .....	4
2.4 RESULTS FUSION.....	5
<b>3. USAGE GUIDE.....</b>	<b>6</b>
3.1 DIM CONTROL AND COMMUNICATION COMMANDS SET.....	7
<b>4. RESULTS FUSION.....</b>	<b>9</b>
4.1 FUSION METHODS.....	9
4.2 FUZZY LOGIC .....	10
4.3 COMBINATION MODES.....	15
<b>5. SOFTWARE STRUCTURE .....</b>	<b>15</b>
5.1 <i>DIM_SLM PROGRAM</i> .....	15
5.2 <i>MATLAB_ENG PROGRAM</i> .....	16
5.3 MATLAB FUNCTIONS .....	17
<b>6. CODE MODULES.....</b>	<b>17</b>
<b>7. BIBLIOGRAPHY .....</b>	<b>18</b>
<b>APPENDIX A: DATA FIELDS FOR DIM SAMPLE PROCESSING .....</b>	<b>20</b>
<b>APPENDIX B: DIM - TSC PROCESSING SCRIPTS .....</b>	<b>22</b>
<b>APPENDIX C: ASCII RESULTS FILE .....</b>	<b>26</b>
<b>APPENDIX D: “C”CODE LISTINGS .....</b>	<b>27</b>
<b>APPENDIX E: MAKEFILE .....</b>	<b>84</b>
<b>APPENDIX F: MATLAB FUNCTION LISTINGS.....</b>	<b>90</b>



## **1. Introduction**

The data interpretation module (DIM) is one of the standard laboratory modules (SLM) used in the automation of PCB sample analysis<sup>5,10</sup>. This module consists of software, which takes as input the raw chromatogram produced by the analytical instrument (gas chromatography system) and produces an estimate of the concentration of specific analytes under investigation. All of the steps in this process are initiated and controlled by the task sequence controller (TSC) via an electronic communications link. The DIM consists of several distinct pieces including the UNIX executable control and communication (CC) program, the UNIX executable MATLAB interface program, and the MATLAB programs which perform the analytical computations.

This document will focus on the components of the DIM related primarily to the interface with the TSC, the control and sequencing of the MATLAB analysis functions, and the algorithms for the combination of results obtained from multiple analytical methods. The document will be organized into a functional description section followed by an instructional usage guide. Technical aspects of the results fusion, software structure, and code modules will be covered in the following sections. The appendix will include an example of the ASCII results file, and code listings.

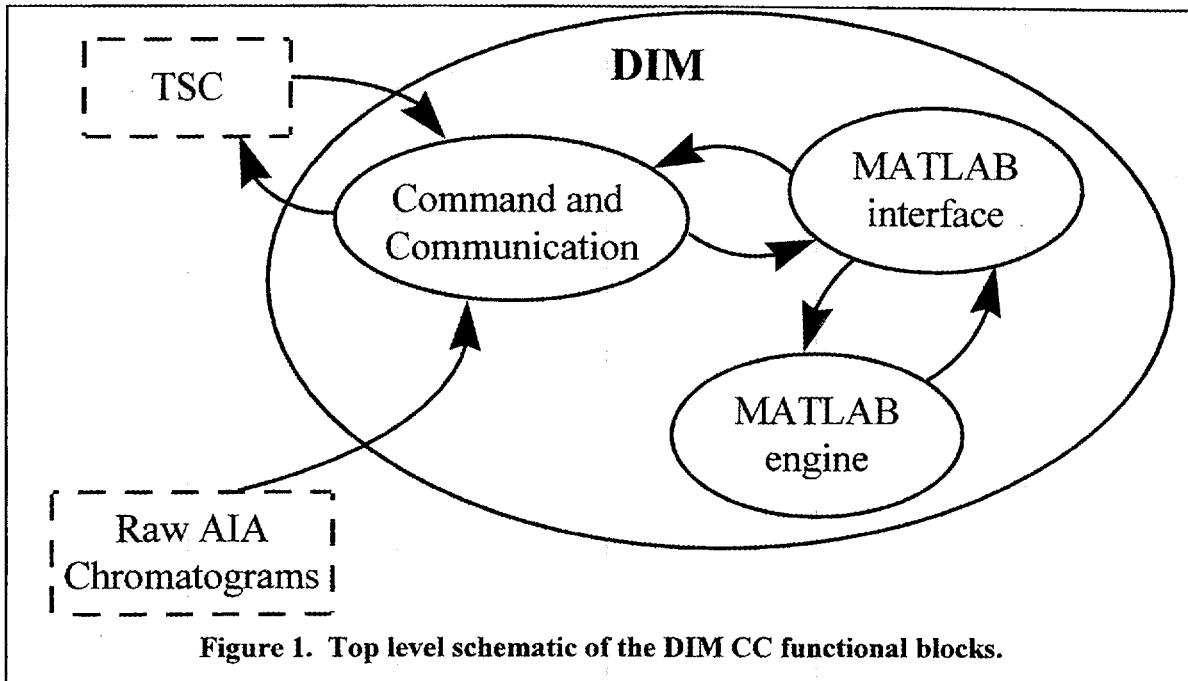
## **2. Functional description**

The primary function of the CC component of the DIM is to translate commands issued from the TSC into the appropriate MATLAB function calls and combine the results from various analytical methods. A modular approach can be taken to meet the required functionality with the primary modules being the TSC-DIM communications interface, DIM-MATLAB interface, data management, and results fusion. Each of these functional modules will be described in the following sections. The schematic in Figure 1 shows a diagram of the functional modules of the DIM CC.

### **2.1 TSC-DIM interface**

The TSC is the master controller of the standard analysis method (SAM) and each SLM must respond to commands issued as part of the SAM processing script. During the analysis of a sample the TSC will instruct the appropriate SLMs to perform their respective functions. The last step in this process is the analysis of the raw chromatogram generated by the analytical instrument module (AIM) to extract chemical knowledge about the sample. The functional requirements of the interface between the TSC and the DIM are listed below.





**Figure 1. Top level schematic of the DIM CC functional blocks.**

1. Establish communication with the TSC. When the DIM CC program starts, the first task is to establish a communications channel and identify itself by exchanging the relevant information about the DIM to the TSC. The SLM interface tool kit will be used as the underlying code to configure and maintain the socket-based connection between the TSC and DIM.
2. Respond to all TSC requests. All commands issued by the TSC must be acknowledged by the DIM, then progress and completion messages must be sent back to the TSC as the requested task is executed. Two levels of commands will be issued by the TSC: laboratory unit operations (LUO) and intra-LUO (ILUO) operations. The LUO commands are queued in a first in, first out (FIFO) buffer and do not interrupt an ongoing command execution while the ILUO commands require an immediate response.
3. Define a common access file system. The DIM needs to be able to copy the raw chromatogram file generated by the AIM to a batch-processing directory. The TSC will supply a filename to the DIM and the DIM will copy the file to the currently defined batch processing directory based on the Analytical Instruments Association (AIA) network common data format (NetCDF) sample type.

## 2.2 DIM-MATLAB interface

The majority of the analytical computations used to convert the raw chromatogram signal to useful chemical knowledge of the sample contents are performed in the MATLAB software environment. This environment is typically accessed via a command line or graphical interface in which the user types commands or makes appropriate user interface selections to execute the desired functions. In the automated processing scenario these commands must be generated by a controlling interface to the MATLAB processing engine. The primary functional requirements of this interface are to establish a message passing queue between the DIM CC and the MATLAB interface program, open a

MATLAB processing engine, generate the appropriate MATLAB function calls and arguments, and parse the return arguments from the MATLAB functions. Each of these tasks will be described in greater detail below.

1. Communications between DIM CC and MATLAB interface. MATLAB provides a mechanism to start a processing engine from a user written executable program. The MATLAB engine function enables text strings to be constructed and passed to the engine in a manner similar to the way the user would type commands at the MATLAB prompt. Because of the ILUO response requirements, the DIM CC must not be blocked by the call to the MATLAB engine. Therefore a standalone executable program with a non-blocking communication link is required to interface between the DIM CC and the MATLAB processing engine.

The communications link between the DIM CC and the MATLAB interface can be a simple message queue. A message queue is a first in, first out (FIFO) queue that has a user defined message structure. A library of support calls exist which enable the opening, formatting, sending, and receiving of messages between two independent executable code modules.

2. Open a MATLAB processing engine. In order to execute the processing algorithms implemented in MATLAB code a MATLAB engine must be opened. A separate program is used to interface with the MATLAB engine using a set of library functions provided by MATLAB. These functions start the MATLAB process and provide a handle for other functions to use in subsequent processing. This program is in a continuous loop, which waits until a message is available on the queue, passes the command string to the MATLAB engine, waits for completion, composes a result message, and sends the message back to the CC.
3. Generation of MATLAB function calls. The interface between the DIM CC and the MATLAB interface must compose a text string containing the necessary function name and required arguments. Based on the command received from the TSC an appropriate MATLAB function will be selected and the arguments extracted from data provided by the TSC. This information will be formatted and put into a message structure to be sent to the MATLAB engine interface program.
4. Parsing return arguments. At the completion of MATLAB functions the return arguments are placed in the MATLAB environment memory and must be retrieved into the interface's memory space. The return variables are then parsed and formatted into a message to be sent to the DIM CC program. After this message has been sent the input message queue is checked and the next MATLAB function request is processed.

### **2.3 Data management**

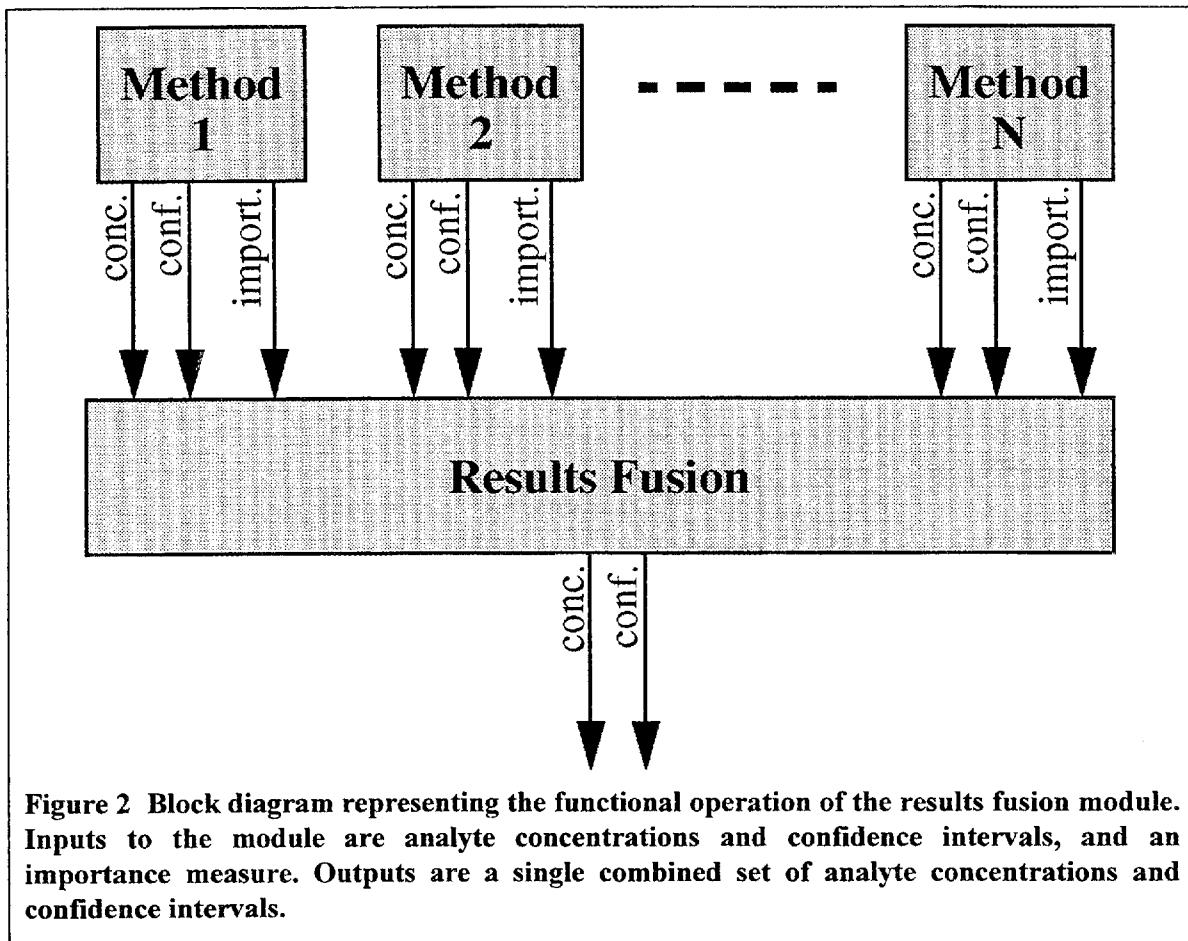
The DIM CC is required to manage both the batch processing specific information and the per sample information during on-line processing. Upon start-up the CC will initialize an internal database of parameters with default values (this state can also be reached by an initialization command). The TSC is responsible for communicating information regarding the current batch processing information prior to any processing of samples. This information will be retained across all samples until a new batch is defined. The TSC will also send sample specific information prior to issuing the processing commands. Finally, sample specific information is generated during the

quality and analysis processing. The details of the data management function are listed below.

1. Establish and maintain DIM CC database. A defined data structure with the required fields, as defined in the appendix section A, "Data fields for DIM sample processing," is generated in memory during the startup of the DIM CC. The TSC will issue commands, which contain information that is stored in the data structure and subsequently used during sample processing. Results from the sample processing will also be stored in the data structure. Checks will be made on the required fields prior to executing MATLAB functions and fields, which change on a sample basis, will be cleared upon completion of each sample's processing.
2. Generation of ASCII results file. At the completion of each sample's processing the sample data structure is appended to an ASCII file located at the top level of the batch-processing directory. This file contains a single line for each sample processed and each field described in appendix A is separated with a blank space. An example of this results file is listed in Appendix C.

#### **2.4 Results Fusion**

The final functional component of the DIM CC is combining the results from several different analytical analysis methods such as principal component regression (PCR)<sup>4</sup>, multiple linear regression (MLR), and artificial neural networks (ANN)<sup>11</sup>. Results from each method report a concentration and confidence interval for each analyte and an overall "importance" measure on how well the reported concentrations account for the original measured signal. This module utilizes both the reported concentrations and the importance measure to combine the results into a single result as shown in the block diagram of Fig. 2. A fuzzy logic based fusion strategy is used to perform the actual combination. The functional tasks of this component include calling the available individual analytical methods and generating the fuzzy logic based weighting factors, generating a single result for each analyte, and writing the results into the sample data structure. A detailed technical description of the results fusion function is given in Section 4.



### 3. Usage Guide

The DIM CC is straightforward to start and there is minimal user interaction once the program is running. To use the DIM CC, the main executable module, *dim\_slm*, must be started at the UNIX command prompt with the appropriate arguments. Once the main module starts, the program automatically starts the module which communicates with the MATLAB engine. The steps listed below describe the individual operations for proper startup of the DIM CC. Once the DIM CC is running and a connection is made with the TSC, it will respond to issued commands. A short summary of the valid commands will be listed below and several example TSC scripts are given in appendix section B, "Example DIM - TSC processing scripts."

Steps to start DIM CC:

1. Set the current working directory to the "general" section of the DIM software distribution, e.g. "cd /caa/dim/general".
2. Set the "DISPLAY" UNIX/ X Windows environment variable to the machine that you desire the graphic displays to be viewed on, e.g. "setenv DISPLAY klatt.rpsd.ornl.gov:0.0".
3. Start the DIM CC application using one of the two following argument lists.

“dim\_slm *slmid* service broadcastport” or “dim\_slm *slmid* service host port”

where *slmid* is the numeric identification of this particular SLM, e.g. 0; *service* is the type of connection desired (either TSC or HCI); *broadcastport* is the numeric value used to locate a service in the broadcast mode (this must match the number used by the service); *host* is the fully qualified host name of the machine the service is running on; and *port* is the numeric value of a specific port on the specified host.

4. To exit the program type “Quit” in the window the *dim\_slm* program was started in. This action will sever all communications links and terminate both the *dim\_slm* and MATLAB interface programs.

After step three the *dim\_slm* program will try to establish a communication link with the requested service and start the program which interfaces with the MATLAB engine. The user will see a connection-established message in the window where the *dim\_slm* program was started. In addition the MATLAB startup graphic will display on the screen of the host specified in the DISPLAY environment variable. If the display host denies permission to display graphics, MATLAB will run without any graphics display (potential source of this problem is not issuing the *xhost* command on the display server). The MATLAB engine may fail to start if the MATLAB executable files are not found in the users search path (PATH environment variable).

Once the DIM CC has started and a communication link is established, the program is ready to accept commands from the TSC and perform the requested action. The set of valid DIM CC commands is listed in the following section.

### **3.1 DIM control and communication commands set**

The commands are listed in bold. Capitalization is not significant. Commands with multiple arguments have the argument options listed in the order they appear in the command string.

1. **Identify** - The DIM will respond with the IDENTITY response.
2. **Initialize** - The DIM will acknowledge and initialize all variables and methods.
3. **Start ACT=Add\_std** - The DIM will execute the code necessary to add a new standard sample to the batch directory hierarchy.
4. **Start ACT=Validate** - The DIM will execute the algorithms necessary to validate the raw GC signal generated by the AIM.
5. **Start ACT=Analyze** - The DIM will execute the algorithms and analytical methods to determine the concentration and respective confidence interval of the sample.
6. **Set PAR=<parameter> VAL=<value>**
  - a. **Comb**
    - i. **Max** - Maximum from all methods
    - ii. **Min** - Minimum from all methods
    - iii. **Avg** - Average results from all methods
    - iv. **Fusion** - Fusion using importance parameters

- v. **PCRR** - No combination, Principal Component Regression method on the raw chromatogram
- vi. **PCRP** - No combination, PCR method on the peak areas
- vii. **MLRR** - No combination, MLR method on the raw chromatogram
- viii. **MLRP** - No combination, MLR method on the peak areas
- ix. **LRP** - No combination, linear regression on peak areas
- x. **NNP** - No combination, Artificial neural network on peak areas
- xi. **SLER** - No combination, Simultaneous Linear equations on the raw chromatogram

**b. Batch\_path**

- i. character string of the pathname to the batch directory, no spaces

**c. AIA\_samp\_type**

- i. standard
- ii. unknown
- iii. control
- iv. blank

**d. AIA\_samp\_id**

- i. character string of the sample id, no spaces

**e. Datafile\_name**

- i. character string of the full path name of the raw chromatogram and peak area file

**f. AIA\_sample\_amt**

- i. character string of the sample amount in mg.

**7. Read PAR=<parameter>** - The following list of DIM parameters can be read by the TSC:

- a. Combination\_mode
- b. Batch\_path
- c. Peak\_file
- d. AIA\_sample\_type
- e. AIA\_sample\_id
- f. Datafile\_name
- g. AIA\_sample\_amt

## Inter Laboratory Unit Operations (ILUO)

- 1. Abort** The DIM will abort current operation and return to the initialized state.
- 2. Status** The DIM will report its current status.
- 3. Reply** The DIM receives information requested by the Query response.

- 4. Bypass**    **TXT=<cmd string>** - The DIM will execute the command string in the MATLAB environment.

## 4. Results Fusion

In many applications involving redundant measurements of a single physical phenomenon the aggregation of the measurements into a more accurate and precise analysis is desired. This type of operation is generally referred to as information or sensor fusion and is especially useful where potential conflicts exist between the measurements or a *priori* information is known about the reliability of information under certain conditions<sup>1,9</sup>. In this application several methods analyze the measured chromatogram time series and each generates an estimate of the analyte concentrations in the unknown sample. Each method has its strengths and weaknesses and the goal of combining the multiple results is to utilize confidence measures reported by each method to control the aggregation. In general the two primary advantages of fusion that are relevant to this application are redundancy of information and complementary information.

A significant contribution of the DIM function is the intelligent combination of the analytical results from several analysis methods using fusion. The DIM CC performs this combination using several techniques including minimum, maximum, average, and fuzzy logic. The primary goal of the fuzzy logic based combination is the incorporation of both heuristic rules describing which methods perform best under certain analyte conditions and the reported importance measure from each method. The following two sections will present several basic fusion methods and a detail description of the fuzzy logic approach to fusion.

### 4.1 Fusion methods

The most basic approach to analytical results integration is the use of a statistical moment such as the mean or weighted mean to combine the reported analytes concentration and variances. This approach is not optimal in a statistical sense but has computational simplicity and few constraints or required prior knowledge of the information being combined<sup>12</sup>. The mean operator can be replaced with other nonlinear operators such as min, max, and, median. A Kalman filter extends this general concept to incorporate the estimated statistical characteristics of the measurements and then generate an optimal filter for the fusion of the low-level sensor readings.

Another class of fusion operators based on probabilistic models includes Bayesian reasoning and evidence theory<sup>3</sup>. With the Bayesian approaches the prior and estimated conditional probability distributions of the measurements are used to reduce uncertainty using the formal statistical combination theorems of Bayes. This approach requires either a large amount of data to generate the probability distributions or a means of reliably estimating the distributions. In this application of results fusion there are not any discrete classes or events to assign probabilities, but rather a linguistic description of conditions

and combination rules. Dempster-Shafer (DS) evidence theory has also been applied to fusing uncertain information using mass functions to represent sensor information and Dempster's rule of combination to combine the information sources<sup>3</sup>. A potential disadvantage of DS is the theory assumes the information sources to be independent which is probably not true for the fusion of different analytical methods applied to the same raw data.

A final approach considered for the fusion of results is fuzzy logic or set theory. Zadeh first proposed fuzzy set theory as a means to represent inexact, incomplete, and uncertain information in a mathematical framework<sup>13</sup>. Fuzzy set theory generalizes the binary valuation of set membership to the real interval [0, 1] and in doing so enables set membership to be represented in degrees. This set theory also includes the traditional union and intersection operators, defined for fuzzy sets, which are used in combining information in this framework. In general this approach has been successful in applications that contain imprecise information and the desire to encapsulate expert knowledge in rule based system<sup>14</sup>. The fuzzy logic approach will be pursued as one possible solution to the task of combining the results from several analytical methods.

## 4.2 Fuzzy logic

Fuzzy logic based results fusion has many advantages in this application including the ability to qualitatively assign set definitions and encapsulate expert knowledge in the rule base. In the following sections the basic definitions of fuzzy logic are given and the specific implementation of results fusion using the underlying framework of fuzzy logic is presented.

### 4.2.1 Definitions

#### Membership functions

A typical way to denote a fuzzy set  $A$  in the universe of discourse  $X$  is with a set of ordered pairs

$$A = \{(x, \mu_A(x)) | x \in X\}, \quad 1$$

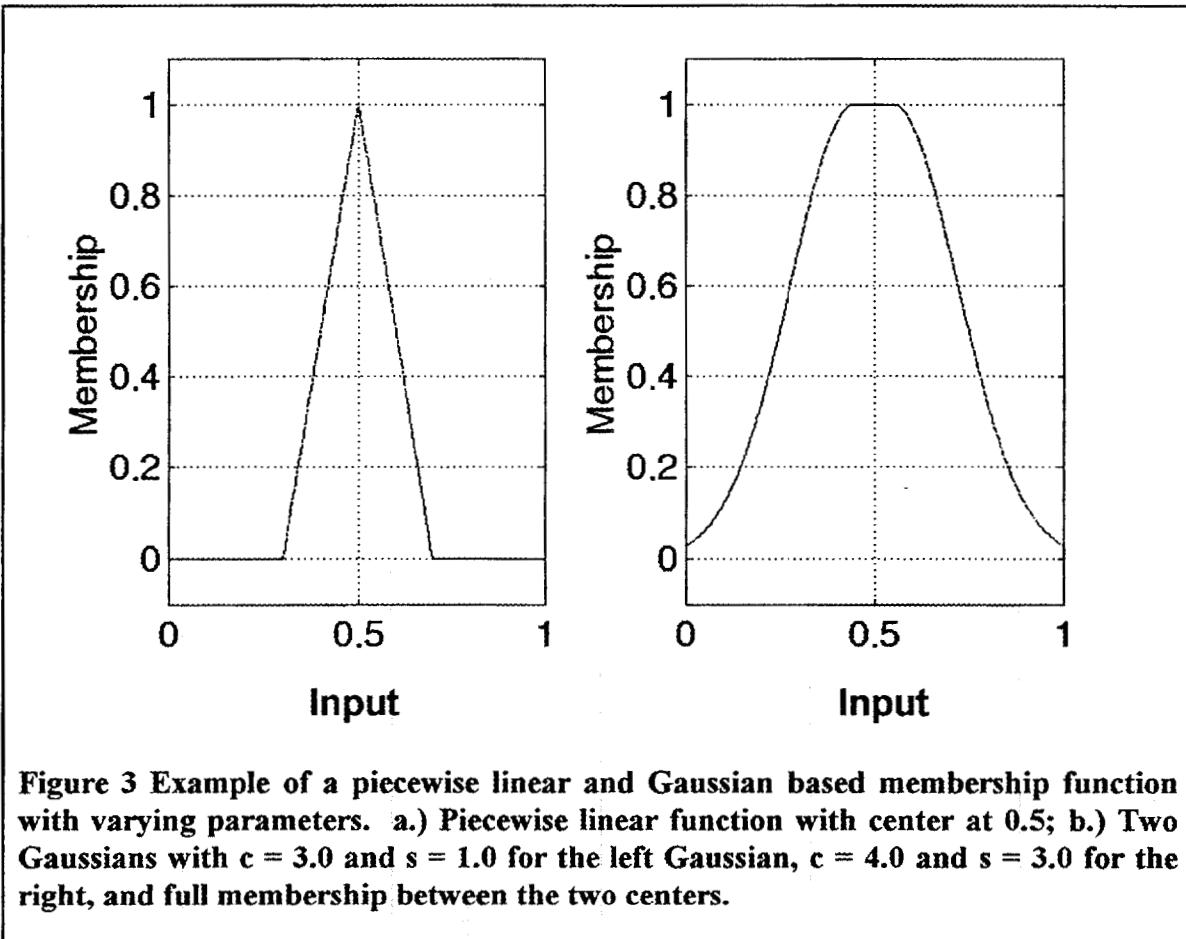
where  $\mu_A(x)$  is the grade of membership or membership function of  $x$  in  $A$  which maps  $X$  to the membership space  $M$ . If  $M$  contains only two elements, 0 and 1,  $A$  is nonfuzzy and  $\mu_A(x)$  is identical to the characteristic function of a nonfuzzy set. As stated in the introduction inexactness can be represented by a fuzzy set. In general three types of inexactness are significant: (1) generality, a concept applies to a variety of situations; (2) ambiguity, a concept describes more than one distinguishable subconcept; (3) and vagueness, a concept does not have precise boundaries. These types of inexactness are represented by fuzzy subsets in the following way: (1) generality, the universe of discourse  $X$  is not just one point; (2) ambiguity, the membership function  $\mu_A(x)$  has

One way to define a membership function is with a continuous standard function such as a Gaussian, sigmoid, or polynomial. An example of such a membership function is given by

$$\mu_A(x; \sigma, c) = e^{\frac{-(x-c)^2}{2\sigma^2}}, \quad 2$$

where  $c$  is the center of the membership function and where the degree of membership is one and  $\sigma$  controls the rate of decreasing membership as  $|x - c|$  increases. The graph in Fig. 3 shows several membership functions based on piecewise linear and functions of the form given in Eq. 2.

In this application the outputs from the analytical methods form the bases of two universes of discourse which will be labeled  $C$  and  $I$  for concentration and importance respectively. The fuzzy sets within the universe  $C$  include present and absent and the sets in the universe  $I$  include low, medium, and high. Membership in each of these sets is determined in a fuzzification procedure, which maps a crisp input value,  $x$ , to a fuzzy membership value,  $\mu_A(x)$  in each set contained in the given universe.



**Figure 3 Example of a piecewise linear and Gaussian based membership function with varying parameters. a.) Piecewise linear function with center at 0.5; b.) Two Gaussians with  $c = 3.0$  and  $s = 1.0$  for the left Gaussian,  $c = 4.0$  and  $s = 3.0$  for the right, and full membership between the two centers.**

### Logical operations

A fundamental set of operations in any set theory is the union, intersection, and negation of sets and their parallel logic operations or, and, and not. In fuzzy set theory these operations are typically defined based on nonlinear operations on the membership values of each set. The theoretical derivations of the operators for intersection and union have been justified by Bellman and Giertz<sup>2</sup> and by Fung and Fu<sup>6</sup>.

Three basic operations of set theory: union, intersection, and complement can be defined for fuzzy sets. Let  $A$  and  $B$  be fuzzy sets of  $X$ . The union of fuzzy sets  $A$  and  $B$  is denoted by  $A \cup B$  and is defined by

$$A \cup B = \int_x \frac{\mu_A(x) \vee \mu_B(x)}{x}, \quad 3$$

where  $\vee$  is the symbol for maximum. The intersection of  $A$  and  $B$  is denoted by  $A \cap B$  and is defined by

$$A \cap B = \int_x \frac{\mu_A(x) \wedge \mu_B(x)}{x}, \quad 4$$

where  $\wedge$  is the symbol for minimum. The complement of  $A$  is denoted by  $\bar{A}$  and is defined by

$$\bar{A} = \int_x \frac{1 - \mu_A(x)}{x} \quad 5$$

These equations reduce to a single point minimum, maximum, and complement for discrete members of the fuzzy sets.

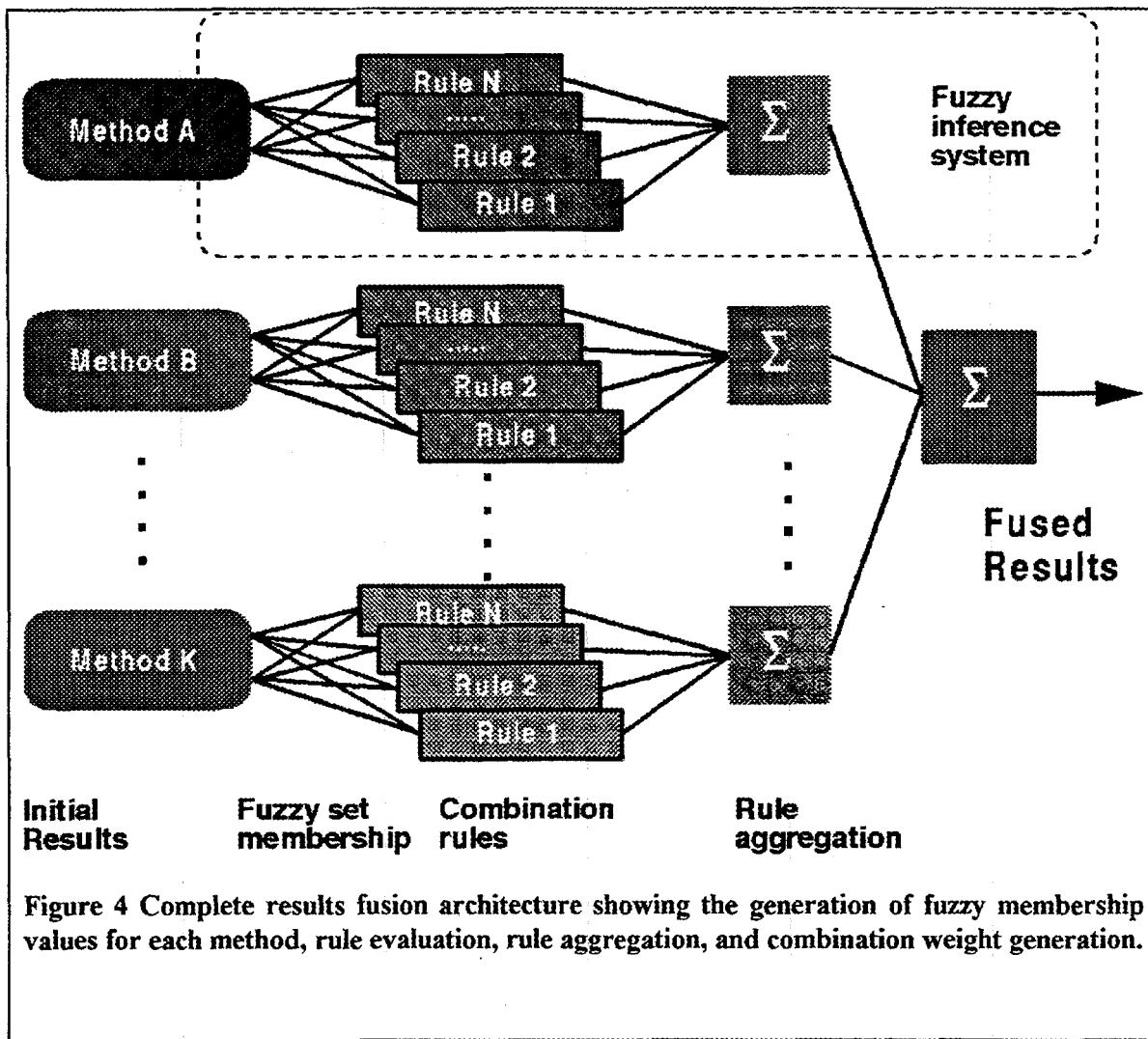
#### 4.2.2 Combination Rules

With these basic definitions the next step in the process of combining results is defining a set of combination rules. These rules state the conditions under which the operators described in the previous section are applied to input values. A typical rule is of the logic form "antecedent then consequent" where the antecedent usually contains the operators listed above and the consequent is membership in another fuzzy set. An example of such a rule is "if concentration is  $A$  and importance is  $B$  then output weight is  $C$ , where concentration, importance, and weight are universes (input/output variables) and  $A$ ,  $B$ , and  $C$  are fuzzy sets. The complete fuzzy combination system consists of the defined set of membership functions and operations, a set of combination rules and an aggregation method for the results of all the rules. The next section will outline how these components combine to accomplish the multiple analytical results integration.

#### 4.2.3 Fusion of Results

The complete process proposed for the combination of the individual results from each analytic method is based on the fuzzy principles described above and will be explained in this section. The schematic shown in Fig. 4 depicts the architecture of the proposed system.

The first step in this process is to run the analytical methods on the raw data and generate a set of output data consisting of the analytes' estimated concentrations, confidence intervals, and overall importance of the results. These results are then input to the fuzzy inference system and the crisp values are mapped to their appropriate fuzzy set using the defined membership functions. All of the rules evaluate the input fuzzy sets and produce a fuzzy output set. The results from each rule are combined into a single fuzzy set using aggregation method such as maximum or sum. A final step in the fuzzy inference process



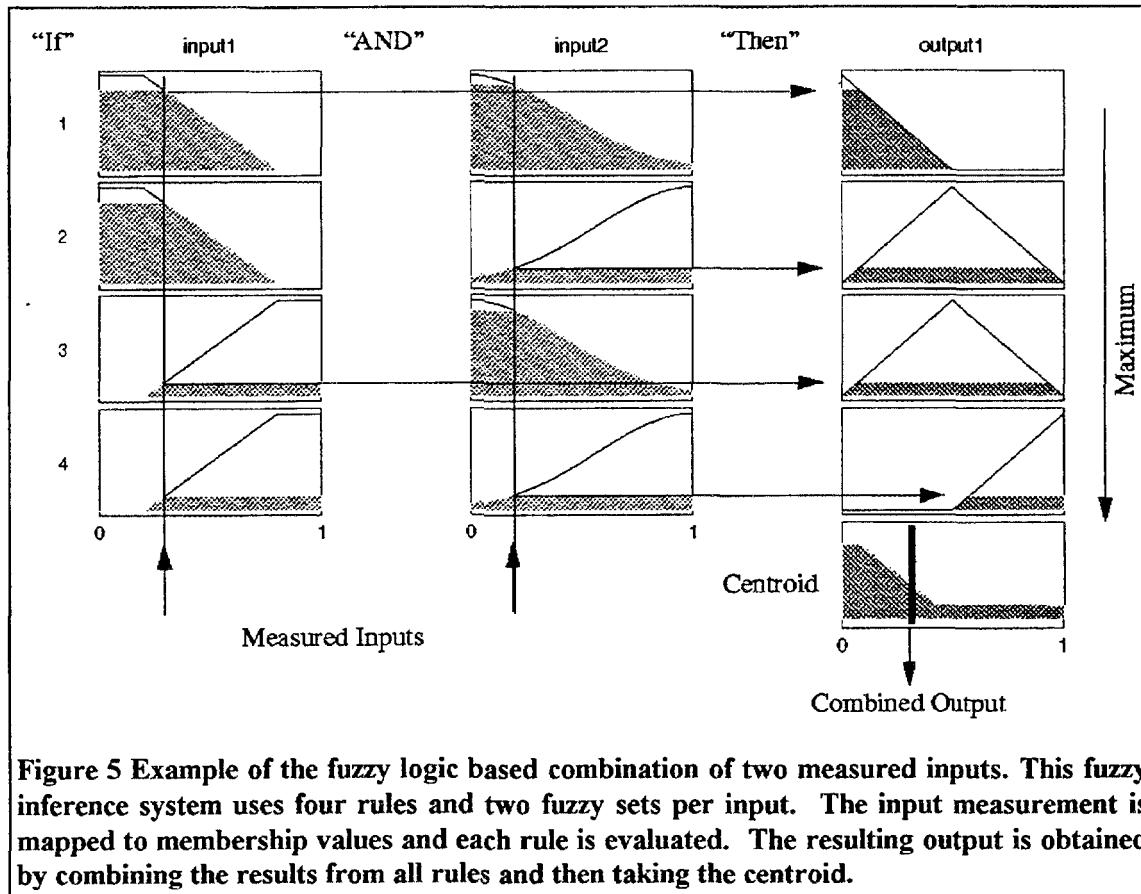
is to defuzzify the output membership function using an operation such as the centroid or area bisector<sup>8</sup>. This process is depicted graphically in Fig. 5.

The output of each fuzzy inference system is a weight factor,  $w_i$ , between 0.0 and 1.0 which is used in a weighed average of the concentration and confidence interval for each analyte. Specifically the concentration is given by

$$conc(anal)_{final} = \frac{\sum_i w_i \cdot conc_i(anal)}{\sum_i w_i}, \quad 6$$

where  $i$  is the set of all analytical methods and  $w_i$  is the weight result derived from the fuzzy system. The confidence interval is combined in a similar manner using

$$conf(anal)_{final} = \sqrt{\frac{\sum_i w_i \cdot (conf_i(anal))^2}{\sum_i w_i}}. \quad 7$$



**Figure 5** Example of the fuzzy logic based combination of two measured inputs. This fuzzy inference system uses four rules and two fuzzy sets per input. The input measurement is mapped to membership values and each rule is evaluated. The resulting output is obtained by combining the results from all rules and then taking the centroid.

Several items must be configured prior to the use of the results fusion including the definition of membership functions and generation of the combination rule set. These are typically defined based on the expert knowledge of the system designer. Some work has been done by Jang on automating the generation of the rule base in a fuzzy system<sup>7</sup>.

### **4.3 Combination modes**

The DIM control and communication (CC) program implements four results combination modes in addition to the ability to select a single analytical method. The combination modes include unweighted mean, minimum, maximum, and fuzzy weighted mean. Each of these combination modes considers the concentration results from each available analytical method. The CC program generates a table of results as each analytical method is run with each row the results from a specific method and columns the concentration and confidence of each analyte and the overall importance measure for that method. This table is then used as the input to the combination module.

The unweighted average combination method generates a resulting analyte concentration by taking the arithmetic mean of each concentration column. Since the confidence interval is a measure similar to the standard deviation, the resulting confidence interval is the root mean square (RMS) of the individual measures. The unweighted average is given by Eq. 6 with  $w_i$  equal to unity for all  $i$  and the confidence interval given by Eq. 7 with  $w_i$  equal to unity.

The minimum and maximum combination modes generate the respective minimum or maximum concentration of all the computed concentrations for a given analyte. The confidence interval is obtained from the corresponding concentration (not the minimum or maximum confidence).

## **5. Software Structure**

As described in Section 2 the primary software modules in the DIM are the TSC-DIM communications interface, DIM-MATLAB interface, and analytical computation/data management. Each of these modules consists of either C/C++ code or MATLAB code. The structure of the DIM software will be described in the following sections for each major functional module. Each section will describe the components, which make up the overall functional module and the layout of the actual code.

### **5.1 *DIM\_SLM* program**

The executable program *dim\_slm* is the primary program of the DIM and it contains several source files. The main program file is “*dim\_slm.c*” and the additional support files are “*dim\_util.c*,” “*dim\_cmds.c*,” and a header file “*dim\_slm.h*.” This program uses the generic CAA communications toolkit library and thus consists of a main program, which calls the library function “*Tkexecutive*.” The remaining code consists of functions that execute under specified TSC commands such as *initialize*, *start*, *read*, etc. One important attribute located in the main section of the code is the forking (or creation) of a child process which starts the program “*matlab\_eng*.” Upon execution of the main program, the program splits and the child process starts the program which interfaces with the MATLAB processing engine and the parent process sets up communication with the “*matlab\_eng*” program and then enters the toolkit executive.

The individual functions that are registered with the toolkit are *diminit*, *dimstart*, *dimset*, *dimread*, *idle*, *bypass*, *intraLUO*, and *error*. Most of the action is initiated via one of the

first four functions and most of the time is spent in the *idle* function waiting for new commands or for commands to complete. A global variable, *slmstate*, is used to enable specific action to be taken within any of the above listed toolkit support functions. The variable can be set to the current action of the DIM when a command is started and cleared when the command completes. A typical execution of a TSC command would proceed in the following sequence:

- (1) the appropriate action function would be called (e.g. *dimstart*),
- (2) within the function a message is generated and sent to the “matlab\_eng” program (if MATLAB processing is required to complete the action),
- (3) the function sets the *slmstate* variable to indicate the processing state,
- (4) the idle function is entered and the message queue is checked to determine if the MATLAB processing has completed,
- (5) if a message is on the queue, the *slmstate* is set to “IDLE” and the complete message is returned to the TSC.

## 5.2 *MATLAB\_ENG* program

The executable program *matlab\_eng* is a C program that interfaces between the *dim\_slm* program and the MATLAB computational engine. This program links these two components in a non-blocking manner so that asynchronous communication can occur between the *dim\_slm* program and the TSC. An additional layer of software was required because the standard library function, which executes MATLAB code, blocks the calling program until the MATLAB code completes.

The *matlab\_eng* program utilizes message passing to receive commands and transmit results from/to the *dim\_slm* program. A MATLAB command string is generated by the *dim\_slm* program and packaged into the body of a message and placed onto the queue. The *matlab\_eng* program continually polls the receive message queue and parses commands when a message is received. A call to the MATLAB engine starts the desired processing. Once the *dim\_slm* program has sent the message it is free to listen for commands from the TSC and return messages from the *matlab\_eng* program. When the MATLAB engine returns, the result is formatted into a message and sent to the *dim\_slm* program.

The *matlab\_eng* program thus runs in a continuous loop of waiting for a message to arrive, passing the message on to MATLAB, waiting for completion, sending the results out in a message. The messaging facility provides the capability for multiple messages to be stored in the queue and processed in a first in first out (FIFO) order.

## 5.3 *MATLAB* functions

The analytical computation and some data management are performed by functions written in MATLAB. This computing environment enables robust and efficient algorithm development for analytical chromatogram processing. As described in the proceeding sections, the MATLAB function calls are formulated by the control and communication

program and passed to the MATLAB environment via the MATLAB interface program. The primary functions include chromatogram preprocessing, QA assessment, analytical methods, and results fusion. MATLAB functions are similar to other procedural languages with a calling format of “[output1, output2, ...output n] = *function name* (input1, input2, ... input n)”. Each of the functions includes a description of the processing performed in the header area of the actual code.

## 6. Code modules

The actual code listing for the “C” programs is included in Appendix D. A listing of the Makefile for the C programs is listed in Appendix E. The MATLAB functions are listed in Appendix F.

## 7. Bibliography

- [1] M. A. Abidi and R. C. Gonzalez, eds. Data Fusion In Robotics and Machine Intelligence, Academic Press, San Diego, CA, 1992.
- [2] R. Bellman and M. Gertz. "On the analytical formalism of the theory of fuzzy sets," Information Sciences, 5:149 - 156, 1973.
- [3] I. Bloch. "Information combination operators for data fusion: A comparative review with classification," IEEE Trans. on Sys., Man and Cybernetics - Part A: Sys. and Humans, 26(1): 52 - 67, 1996.
- [4] J. W. Elling, L. N. Klatt, and W. P. Unruh. "Automated Data Interpretation in an Automated Environmental Laboratory," Laboratory Robotics and Automation, 6(2): 73 - 78, April 1994.
- [5] T. H. Erkkila, R. M. Hollen, and T. J. Beugelsdijk. "The Standard Laboratory Module: An Integrated Approach to Standardization in the Analytical Laboratory," Laboratory Robotics and Automation, 6(2): 57 - 64, April 1994.
- [6] Fung, L. W. and K. S. Fu. "The K<sup>th</sup> optimal policy algorithm for decision-making in fuzzy environments." Identification and System Parameter Estimation (P. Eykhoff, Ed.). North Holland, pp. 1025-1059, 1974.
- [7] J.-S. R. Jang. "ANFIS Adaptive-Network-based Fuzzy Inference System," IEEE Trans. on Systems, Man, and Cybernetics, 23(3):665 - 685, 1993.
- [8] A. Kandel. Fuzzy mathematical techniques with applications, Addison-Wesley, Reading, MA, 1986.
- [9] L. F. Pau. "Sensor data fusion," Journal of Intelligent and Robotic Systems, 1: 103 - 116, 1988.
- [10] F. A. Settle, Jr., R. Hollen, and L. W. Yarbrough. "The Contaminant Analysis Automation Project," American Laboratory, April 1995.
- [11] M. A. Williams. "Application of artificial neural networks in the quantitative analysis of gas chromatograms". M.S. Thesis, University of Tennessee, Knoxville, TN, May 1996.
- [12] R. R. Yager. "A general approach to the fusion of imprecise information," Intl. J. of Intelligent Systems, 12(1): 1 - 29, 1997.

- [13] L. A. Zadeh. " Fuzzy Sets and Applications: Selected Papers by L. A. Zadeh. John Wiley & Sons, New York, 1987.
- [14] H. J. Zimmermann. Fuzzy Set Theory and Its Applications, 2nd ed. Kluwer Academic, Boston, 1991.

## **Appendix A: Data fields for DIM sample processing**

Field Number	Contents
1.	Sample ID that is entered by the chemist with the HCl and transferred to the DIM by the TSC.
1.	Sample type that is entered by the chemist with the HCl and transferred to the DIM by the TSC. Legal sample types are: standard, unknown, control, and blank.
2.	Sample amount that is entered by the chemist with the HCl and transferred to the DIM by the TSC. The units on the sample amount are milligrams.
3.	Filename of the AIA (netCDF) formatted data file for the sample. This filename will be passed to the DIM by the TSC with the extension ".CDF" or ".cdf". The DIM control software will create a second name using the original prefix with ".d" replacing the ".CDF" or ".cdf" extension. This resulting name will point to the subdirectory containing all the detailed information about the data processing applied to the sample.
4.	The prefix of the filename of the calibration set information file. The DIM control software will obtain this filename during execution of the DIM. This field is included because of the probability of reanalyzing the sample(s) using a different calibration set information file.
5.	Result of the signal on-scale QA evaluation. A value of 1 will denote the evaluation passed; a value of -1 will denote the evaluation failed.
6.	Result of the retention time QA evaluation. A value of 1 will denote the evaluation passed; a value of -1 will denote the evaluation failed.
7.	Result of the daily calibration standard QA check. A value of 1 will denote the evaluation passed, a value of -1 will denote the evaluation failed, and a value of 0 will denote this QA evaluation is not applicable to this sample.
8.	The type of combination mode used in the result combination fuzzy logic system. The method of result combination will be fusion based upon the "importance" parameters.
9.	The known amount of the surrogate standard in the sample in units of milligram. (The current contents of this field will be N.I., which means this functionality has not been implemented.)
10.	The calculated recovery of the surrogate standard in the sample, expressed as a percent. (The current contents of this field will be N.I., which means this functionality has not been implemented.)
11.	Name of analyte(1).
12.	Amount of analyte(1) in the sample with units of ppm.
13.	Confidence interval on the amount of analyte(1) with units of ppm.
14.	Name of analyte(2).
15.	Amount of analyte(2) in the sample with units of ppm.
16.	Confidence interval on the amount of analyte(2) with units of ppm.
17.	Name of analyte(3).

18. Amount of analyte(3) in the sample with units of ppm.
19. Confidence interval on the amount of analyte(3) with units of ppm.

## **Appendix B: DIM - TSC processing scripts**

### **1. Introduction**

The following scripts are examples of possible sample processing sequences to be performed by the DIM SLM. Examples will be given for a standard sample flow, for a change in batch, and for several error conditions. Please refer to the “PCB DIM Command syntax” document for details on the actual commands.

### **2. Initialization sequence**

This operation would need to be performed once after the DIM SLM task is started. It may be run at any other time to bring the DIM SLM to a known default state.

#### TSC Command

INITIALIZE

SET PAR=COMB VAL=FUSION

SET PAR=BATCH\_PATH VAL=/caa/data/ornl2

#### DIM Command

000 ACK TO=20

100 COMPLETE RTN=0 TXT=“Initialization Complete”

000 ACK TO=5

100 COMPLETE RTN=0 TXT=“set comb complete”

000 ACK TO=5

100 COMPLETE RTN=0 TXT=“set batch\_path complete”

### **3. Batch processing of samples**

These operations would be performed during the routine operation of processing samples with the SAM system

#### TSC Command

SET PAR=AIA\_SAMP\_TYPE VAL=UNKNOWN

SET PAR=AIA\_SAMP\_ID VAL=C0184678

SET PAR=AIA\_SAMP\_AMT VAL="50.5 mg"

SET PAR=DATAFILE\_NAME VAL=/tmp/data/file1

START ACT=VALIDATE

START ACT=ANALYZE

SET PAR=AIA\_SAMP\_TYPE VAL=UNKNOWN

SET PAR=AIA\_SAMP\_ID VAL=C0184679

SET PAR=AIA\_SAMP\_AMT VAL=46.5

SET PAR=DATAFILE\_NAME VAL=/tmp/data/file2

#### DIM Command

000 ACK TO=5

100 COMPLETE RTN=0 TXT=“set aia\_samp\_type complete”

000 ACK TO=5

100 COMPLETE RTN=0 TXT=“set aia\_samp\_id complete”

000 ACK TO=5

100 COMPLETE RTN=0 TXT=“set aia\_samp\_amt complete”

000 ACK TO=5

100 COMPLETE RTN=0 TXT=“set datafile\_name complete”

000 ACK TO=30

200 PROGRESS TO=30

200 PROGRESS TO=30

100 COMPLETE RTN=0 TXT=“validation complete”

000 ACK TO=60

200 PROGRESS TO=60

200 PROGRESS TO=60

100 COMPLETE RTN=0 TXT=“analyze complete”

000 ACK TO=5

100 COMPLETE RTN=0 TXT=“set aia\_samp\_type complete”

000 ACK TO=5

100 COMPLETE RTN=0 TXT=“set aia\_samp\_id complete”

000 ACK TO=5

100 COMPLETE RTN=0 TXT=“set aia\_samp\_amt complete”

000 ACK TO=5

100 COMPLETE RTN=0 TXT=“set datafile\_name complete”

START ACT=VALIDATE	000 ACK TO=30 200 PROGRESS TO=30 200 PROGRESS TO=30 100 COMPLETE RTN=0 TXT="validation complete"
START ACT=ANALYZE	000 ACK TO=60 200 PROGRESS TO=60 200 PROGRESS TO=60 100 COMPLETE RTN=0 TXT="analyze complete"

.....

#### 4. New Batch processing of samples

These operations would be performed immediately after the change from one batch to another.

##### TSC Command

SET PAR=BATCH\_PATH VAL=/caa/data/ornl2  
SET PAR=AIA\_INJ\_VOL VAL=0.4  
SET PAR=AIA\_SAMP\_TYPE VAL=UNKNOWN  
SET PAR=AIA\_SAMP\_ID VAL=C0184679  
SET PAR=AIA\_SAMP\_AMT VAL=25.9  
SET PAR=DATAFILE\_NAME VAL=/tmp/data/file3

##### DIM Command

000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set batch\_path complete"  
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set aia\_inj\_vol complete"  
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set aia\_samp\_type complete"  
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set aia\_samp\_id complete"  
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set aia\_samp\_amt complete"  
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set datafile\_name complete"

START ACT=VALIDATE

000 ACK TO=30  
200 PROGRESS TO=30  
200 PROGRESS TO=30  
100 COMPLETE RTN=0 TXT="validation complete"  
000 ACK TO=60  
200 PROGRESS TO=60  
200 PROGRESS TO=60  
100 COMPLETE RTN=0 TXT="analyze complete"

START ACT=ANALYZE

## 5. Addition of standard to batch directory

These operations would be performed whenever a new standard is added to the calibration set.

### TSC Command

```
SET PAR=BATCH_PATH VAL=/caa/data/ornl2  
SET PAR=AIA_INJ_VOL VAL=0.4  
SET PAR=AIA_SAMP_TYPE VAL=STANDARD  
SET PAR=AIA_SAMP_ID VAL=C0184679  
SET PAR=AIA_SAMP_AMT VAL=0.004  
SET PAR=DATAFILE_NAME VAL=/tmp/data/file4  
START ACT=ADD_STD  
  
SET PAR=AIA_SAMP_TYPE VAL=STANDARD  
SET PAR=AIA_SAMP_ID VAL=C0184662  
SET PAR=AIA_SAMP_AMT VAL=0.004  
SET PAR=DATAFILE_NAME VAL=/tmp/data/file5  
START ACT=ADD_STD
```

....  
.....

(at the end of this cycle - 15 standards, the operator would be required to enter the DIM GUI and build the models for each method)

### DIM Command

```
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set batch_path complete"  
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set aia_inj_vol complete"  
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set aia_samp_type complete"  
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set aia_samp_id complete"  
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set aia_samp_amt complete"  
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set datafile_name complete"  
000 ACK TO=30  
200 PROGRESS TO=30  
200 PROGRESS TO=30  
100 COMPLETE RTN=0 TXT="Add Standard complete"  
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set aia_samp_type complete"  
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set aia_samp_id complete"  
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set aia_samp_amt complete"  
000 ACK TO=5  
100 COMPLETE RTN=0 TXT="set datafile_name complete"  
000 ACK TO=30  
200 PROGRESS TO=30  
200 PROGRESS TO=30  
100 COMPLETE RTN=0 TXT="Add Standard complete"
```

## 6. Error conditions in Validation

This series of operations might be seen if the validation procedure failed. If this return was seen all further processing by the DIM and GC should cease.

### TSC Command

START ACT=VALIDATE

### DIM Command

000 ACK TO=30

200 PROGRESS TO=30

200 PROGRESS TO=30

100 COMPLETE RTN=-1 TXT="Retention markers out of range,  
check GC"

*... The TSC would issue commands to the give the operator the option to terminate processing of samples on the GC  
.... Once corrective action had been taken, the DIM would start a new batch*

SET PAR=BATCH\_PATH VAL=/caa/data/ornl2

000 ACK TO=5

100 COMPLETE RTN=0 TXT="set batch\_path complete"

SET PAR=AIA\_INJ\_VOL VAL=0.4

000 ACK TO=5

100 COMPLETE RTN=0 TXT="set AIA\_INJ\_VOL complete"

SET PAR=AIA\_SAMP\_TYPE VAL=STANDARD

000 ACK TO=5

100 COMPLETE RTN=0 TXT="set AIA\_SAMP\_TYPE complete"

SET PAR=AIA\_SAMP\_ID VAL=C0184679

000 ACK TO=5

100 COMPLETE RTN=0 TXT="set AIA\_SAMP\_ID complete"

SET PAR=AIA\_SAMP\_AMT VAL=25.9

000 ACK TO=5

100 COMPLETE RTN=0 TXT="set AIA\_SAMP\_AMT complete"

SET PAR=DATAFILE\_NAME VAL=/tmp/data/file3

000 ACK TO=5

100 COMPLETE RTN=0 TXT="set DATAFILE\_NAME complete"

## Appendix C: ASCII results file

062502 unknown 1.000e+03 /mnt/CAA/output/062502.cdf 425460 1 1 0 3 1.600e-01 1.048e02 1242 4.024e-01 2.987e-02 1254 1.434e-01 4.658e-02 1260 9.573e-03 3.520e-02  
062503 unknown 1.000e+03 /mnt/CAA/output/062503.cdf 425460 1 1 0 3 1.600e-01 1.067e+02 1242 1.207e-02 2.018e-02 1254 3.865e-01 3.728e-02 1260 9.015e-03 3.543e-02  
062504 unknown 1.000e+03 /mnt/CAA/output/062504.cdf 425460 1 1 0 3 1.600e-01 1.018e+02 1242 -4.555e-03 1.329e-02 1254 1.455e-02 2.392e-02 1260 3.559e-01 3.295e-02  
062721 unknown 1.000e+03 /mnt/CAA/output/062721.cdf 425460 1 1 0 3 7.200e-02 1.153e+02 1242 -1.207e-02 9.304e-03 1254 2.215e-02 1.744e-02 1260 4.300e-02 9.933e-03  
072309 unknown 1.000e+03 /mnt/CAA/output/072309.cdf 425460 1 1 0 3 1.600e-01 1.748e+01 1242 4.723e-02 3.456e-02 1254 7.652e-02 4.867e-02 1260 -1.027e-02 3.222e-02  
072503 unknown 1.000e+03 /mnt/CAA/output/072503.cdf 425460 1 1 0 3 1.600e-01 4.180e+01 1242 1.357e-01 3.481e-02 1254 2.172e-01 4.748e-02 1260 8.249e-02 3.649e-02  
072505 unknown 1.000e+03 /mnt/CAA/output/072505.cdf 425460 1 1 0 3 1.600e-01 3.692e+01 1242 1.003e-01 3.366e-02 1254 1.887e-01 4.079e-02 1260 3.572e-02 3.965e-02  
072506 unknown 1.000e+03 /mnt/CAA/output/072506.cdf 425460 1 1 0 3 1.600e-01 3.433e+01 1242 1.224e-01 3.399e-02 1254 2.188e-01 4.165e-02 1260 4.633e-02 6.267e-02  
072507 unknown 1.000e+03 /mnt/CAA/output/072507.cdf 425460 1 1 0 3 1.600e-01 1.443e+01 1242 7.141e-02 3.735e-02 1254 1.149e-01 4.460e-02 1260 3.049e-02 6.064e-02  
072508 unknown 1.000e+03 /mnt/CAA/output/072508.cdf 425460 1 1 0 3 1.600e-01 1.452e+01 1242 3.918e-02 3.331e-02 1254 8.874e-02 4.209e-02 1260 1.708e-02 2.856e-02

## Appendix D: "C"Code listings

### aia\_poke.c

```
/*
int aia_poke(char *filename, char *type_val, float amt_val)
```

Inputs:

- filename - a string of the pathname to the netCDF AIA file in which to poke the type and amount volume data.
- type\_val - an ASCII string value indicating the sample type. This will be poked into the GLOBAL attribute "sample\_type".
- amt\_val - a floating point value representing the amount of sample in mg

Outputs:

- Error code representing the status of the function call, negative value indicates failure, zero success

Compile with ANSI C compiler.

Modifications by MAH 6/4/96 - changed sample\_amount to float argument (was char)  
added int error code return with associated  
checks

Modifications by MAH 8/1/96 - changed to aia\_poke, removed injection volumn

```
*/
#define NO_SYSTEM_XDR_INCLUDES
#define NO_GETPID
#define USE_XDRNCSTDIO

#include <stdio.h>
#include <string.h>
#include <stdlib.h>    /* Needed for strtod() and malloc() */
#include "array.c"
#include "attr.c"
#include "cdf.c"
#include "dim.c"
#include "error.c"
#include "file.c"
#include "globdef.c"
#include "iarray.c"
#include "putget.c"
#include "putgetg.c"
#include "sharray.c"
#include "string.c"
#include "var.c"

#include <sys/types.h>
```

```

#include <types.h>
#include <xdr.h>
#include "xdr.c"
#include "xdrstdio.c"
#include "xdrfloat.c"

int aia_poke(const char *cdfname, char *type_val, float amt_val)
{
    int cdfid;      /* Declare file ID */
    int len;        /* Declare length variable */
    int ecode;      /* return error code */
    double *amount; /* Declare pointer to amount value */

    cdfid=ncopen(cdfname,NC_WRITE); /* Open netCDF file and assign ID#*/
    if(cdfid == -1) {             /* Open error */
        fprintf(stderr, "Could not execute ncopen (aia_poke) on file: %s.\n");
        return(-1);
    }
    ecode = ncredef(cdfid);       /* Open in define mode */
    if(ecode == -1) {             /* Check return code */
        fprintf(stderr, "Could not execute ncredef (aia_poke).\n");
        return(ecode);
    }

    ecode = ncattput(cdfid,NC_GLOBAL,"sample_type",NC_CHAR,\n
                     strlen(type_val)+1,type_val);

    if(ecode == -1) {             /* Check return code */
        fprintf(stderr, "Could not execute ncattput, samp_type (aia_poke).\n");
        return(ecode);
    }

    len=1;          /* Number of double values*/
    amount = malloc(sizeof(double)); /* Allocate space for double */
    if(amount == (double *) NULL) {
        fprintf(stderr, "Could not malloc memory (aia_poke).\n");
        return(-1);
    }
    if(amt_val != 0.0){
        *amount = (double)amt_val;
        fprintf(stderr,"Double sample amount value:%11g\n",*amount);
        ecode = ncattput(cdfid,NC_GLOBAL,"sample_amount",NC_DOUBLE,len,amount);
        if(ecode == -1) { /* Check return code */
            fprintf(stderr, "Could not execute ncattput, samp_amt (aia_poke).\n");
            return(ecode);
        }
    }
}

```

```

ecode = ncendef(cdfid);      /* End define mode */
if(ecode == -1) {           /* Check return code */
    fprintf(stderr, "Could not execute ncended (aia_poke).\n");
    return(ecode);
}

ncclose(cdfid);            /* Close netCDF file*/

free(amount);

return(0);
}
dim_cmds.c
#include "dim_slm.h"

int init_cmd( Dim_stateP stateP)
{
    char *cptr;
    char complete[BUFSIZ+1];
    char errmsg[BUFSIZ+1];
    char ack[40];
    int i,j;

#ifndef DEBUG
    fprintf(stderr, "Initialize Command\n");
#endif

/* code to initialize state of DIM follows */
    init_state(stateP);
    print_state(stateP);

    return(0);
}

int read_cmd( char *buf, int server, Dim_stateP stateP)
{
    char *cptr, *valcptr;
    char complete[BUFSIZ+1];
    char errmsg[BUFSIZ+1];
    char ack[40];

#ifndef DEBUG

```

```

        fprintf(stderr, "Read Command\n");
#endif

        if( (cptr = strtok( buf, " ") ) != NULL ) {
/* first thing to do is acknowledge command */
        sprintf(ack, "000 ACK TO=10 KEY=%s\0",buf);
        CMsendMessage( server, ack);
/* code to read state of DIM follows */
        if((cptr=strtok(NULL," ")) != (char *)NULL) { /* gets parameter to set*/
            sprintf(stderr,"Read parameter: %s\n",cptr);
            if(read_state(stateP, cptr, errmsg) != 0) {
                sprintf(errmsg, "Error in Read");
                sprintf(complete, "100 COMPLETE RTN=%d KEY=%s TXT=%s", -1, buf, errmsg);
                CMsendMessage( server, complete);
            } else {
                sprintf(errmsg, "Read sucessful");
                sprintf(complete, "100 COMPLETE RTN=%d KEY=%s TXT=%s", 0, buf, errmsg);
                CMsendMessage( server, complete);
            }
        } else {
            sprintf(errmsg, "No Read parameter");
            sprintf(complete, "100 COMPLETE RTN=%d KEY=%s TXT=%s", -1, buf, errmsg);
            CMsendMessage( server, complete);
        }
    } else { /* invalid command */
        sprintf(errmsg, "Invalid command %s", buf);
        sprintf(complete, "100 COMPLETE RTN=%d KEY=%s TXT=%s", -1, buf, errmsg);
        CMsendMessage( server, complete);
    }
}

```

```

int start_val_cmd( char *buf, int server)
{
    char *cptr;
    char complete[BUFSIZ+1];
    char errmsg[BUFSIZ+1];
    char ack[40];

#ifndef DEBUG
    fprintf(stderr, "Start Validate Command\n");
#endif

    if( (cptr = strtok( buf, " ") ) != NULL ) {
/* first thing to do is acknowledge command */

```

```

        sprintf(ack, "000 ACK TO=30 KEY=%s\0",buf);
        CMsendMessage( server, ack);
/* code to validate chromatograms follows */

        sprintf(complete, "100 COMPLETE RTN=%d KEY=%s TXT=%s", 0, buf, errmsg);
        CMsendMessage( server, complete);

    } else { /* invalid command */
        sprintf(errmsg, "Invalid command %s", buf);
        sprintf(complete, "100 COMPLETE RTN=%d KEY=%s TXT=%s", -1, buf, errmsg);
        CMsendMessage( server, complete);
    }
}

int start_ana_cmd( char *buf, int server)
{
    char *cptr;
    char complete[BUFSIZ+1];
    char errmsg[BUFSIZ+1];
    char ack[40];

#ifndef DEBUG
    fprintf(stderr, "Start Analyze Command\n");
#endif

    if( (cptr = strtok( buf, " " )) != NULL ) {
/* first thing to do is acknowledge command */
        sprintf(ack, "000 ACK TO=30 KEY=%s\0",buf);
        CMsendMessage( server, ack);
/* code to validate chromatograms follows */

    } else { /* invalid command */
        sprintf(errmsg, "Invalid command %s", buf);
        sprintf(complete, "100 COMPLETE RTN=%d KEY=%s TXT=%s", -1, buf, errmsg);
        CMsendMessage( server, complete);
    }
}

```

```

int abort_cmd( char *buf, int server, Dim_stateP stateP)
{
    char *cptr;
    char complete[BUFSIZ+1];
    char errmsg[BUFSIZ+1];
    char ack[40];

#define DEBUG
    fprintf(stderr, "Abort Command\n");
#endif

    if( (cptr = strtok( buf, " ") ) != NULL ) {
        /* first thing to do is acknowledge command */
        sprintf(ack, "000 ACK TO=10 KEY=%s\0",buf);
        CMsendMessage( server, ack);
        /* code to initialize state of DIM follows */
        fprintf(stderr, "Aborting operation and re-initializing\n");
        init_state(stateP);
        print_state(stateP);
        sprintf(errmsg, "Abort complete");
        sprintf(complete, "100 COMPLETE RTN=%d KEY=%s TXT=%s", 0, buf, errmsg);
        CMsendMessage( server, complete);
    } else { /* invalid command */
        sprintf(errmsg, "Invalid command %s", buf);
        sprintf(complete, "100 COMPLETE RTN=%d KEY=%s TXT=%s", -1, buf, errmsg);
        CMsendMessage( server, complete);
    }
}

int status_cmd( char *buf, int server, Dim_stateP stateP)
{
    char *cptr;
    char complete[BUFSIZ+1];
    char errmsg[BUFSIZ+1];
    char ack[40];

#define DEBUG
    fprintf(stderr, "Status Command\n");
#endif

    if( (cptr = strtok( buf, " ") ) != NULL ) {
        /* first thing to do is acknowledge command */
        sprintf(ack, "001 IACK RTN=0 KEY=%s\0",buf);
        CMsendMessage( server, ack);

```

```

/* code to initialize state of DIM follows */
    print_state(stateP);
    sprintf(errmsg, "Status complete");
    sprintf(complete, "100 COMPLETE RTN=%d KEY=%s TXT=%s", 0, buf, errmsg);
/*    CMsendMessage( server, complete); */
} else { /* invalid command */
    sprintf(errmsg, "Invalid command %s", buf);
    sprintf(complete, "100 COMPLETE RTN=%d KEY=%s TXT=%s", -1, buf, errmsg);
/*    CMsendMessage( server, complete); */
}
}

```

### **dim\_slm.c**

```

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
#include <sys/types.h>
#include <sys/IPC.h>
#include <sys/msg.h>
#include <sys/shm.h>
#include <signal.h>
#define CMLIB
#include "tkapi.h"
#include "dim_slm.h"

int getstdin( char *buffer );
int processkeys( char *buffer);

/* command call back prototypes */

int diminit( int chan );
int dimstart( int chan );
int dimset( int chan );
int dimread( int chan );
int comStatus( int type, char *message );

struct TKcmd slmCmds[] = {
    { "initialize", diminit },
    { "start", dimstart },
    { "set", dimset },
    { "read", dimread },
    { NULL, NULL } };

int slmstate=IDLE;
time_t slmstop;

```

```

time_t nextprog;
int extime;
int exlevel;
Dim_state state;
Dim_stateP stateP;

int      tmsqid; /* returned send message queue id */
int      rmsqid; /* returned receive message queue id */
int      shmid;  /* returned shared memory id */

Engine *ep;
char *mat_output;

/* initialize variables */

char *datatosend = "This is suppose to be data";

int diminit( int chan )
{
    int i;
    int ret;
    char errtxt[LBUFF_SIZE+1];

    TKAck( 5, -1);

    ret = init_state(stateP);

    if (ret == 0) {
        ret = print_state(stateP);
    }

    slmstate = IDLE;

    if(ret < 0) { /* error condition */
        dim_err_code(ret, errtxt);
        TKComplete( 0, errtxt, -1 );
    } else {
        strcpy(errtxt,"Initialization successful");
        TKComplete( 0, errtxt, -1 );
    }

    return TK_CONTINUE;
}

int dimstart( int chan )
{

```

```

int ret;
char *action;
char errtxt[LBUFF_SIZE+1];
char cmdstring[LBUFF_SIZE+1];

TKAck( 120, -1);

ret = 0;

if(!TKGetString("act",&action)) { /* error return */
    return TK_CONTINUE;
}

/* check for entries in the required fields of the state table */

if((ret = check_state(stateP)) < 0) { /* error condition */
    dim_err_code(ret, errtxt);
    fprintf(stderr,"%s.\n",errtxt);
    ret = -1;
    TKComplete( ret, errtxt, -1 );
    return TK_CONTINUE;
}

/* perform action */
fprintf(stderr,"Action code:%s.\n",action);
if(strcasecmp(action, "add_std") == 0) {
    slmstate = ADDSTD;
    fprintf(stderr,"Start add standard\n");
    sprintf(errtxt, "Successful %s ", action);
    slmstate = IDLE;

} else if (strcasecmp(action, "validate") == 0) {
    slmstate = VALIDATE;
    fprintf(stderr,"Start validate \n");
    if((ret = validate(stateP)) < 0) { /* error condition */
        strcat(errtxt, decipher_err_codes(ret,1));
        strcat(errtxt, decipher_err_codes(ret,2));
        fprintf(stderr,"%s.\n",errtxt);
    } else {
        sprintf(errtxt, "Successful %s ", action);
    }
} else if (strcasecmp(action, "analyze") == 0) {
    slmstate = ANALYZE;
    fprintf(stderr,"Start analyze \n");
    if((ret = analyze(stateP)) < 0) { /* error condition */
        strcat(errtxt, decipher_err_codes(ret,1));
        strcat(errtxt, decipher_err_codes(ret,2));
        fprintf(stderr,"%s.\n",errtxt);
    }
}

```

```

    } else {
        sprintf(errtxt, "Successful %s ", action);
    }
    sprintf(errtxt, "Successful %s ", action);
} else {
    fprintf(stderr,"Bad action code \n");
    dim_err_code(BAD_ARG, errtxt);
    ret = -1;
}

if (slmstate == IDLE) {
    TKComplete( ret, errtxt, -1 );
}

return TK_CONTINUE;
}

int dimset( int chan )
{
    char *par, *val;
    char errtxt[LBUFF_SIZE+1];
    int rtn_code;

    TKAck( 30, -1);

    if(!TKGetString("par",&par)) { /* error return */
        fprintf(stderr,"Error in Set,Parameter code:empty.\n");
        return TK_CONTINUE;
    }
    if(!TKGetString("val",&val)) { /* error return */
        fprintf(stderr,"Error in Set,Parameter code:%s, value:empty.\n",par);
        return TK_CONTINUE;
    }
    rtn_code = set_state(stateP, par, val);
    if(rtn_code < 0) {
        fprintf(stderr,"Error in Set,Parameter code:%s, value:%s.\n",par,val);
        sprintf(errtxt,"Error in Set,Parameter code:%s, value:%s.",par,val);
        rtn_code = -1;
    } else if (rtn_code >0) { /* warning */
        fprintf(stderr,"Warning in Set,Parameter code:%s, value:%s.\n",par,val);
        print_state(stateP);
        sprintf(errtxt, "Set %s = %s complete with warning!",par,val);
        rtn_code = 1;
    } else {
        fprintf(stderr,"Set %s = %s\n", par, val);
        print_state(stateP);
        sprintf(errtxt, "Set %s = %s complete",par,val);
        rtn_code = 0;
    }
}

```

```

    }

    if (slmstate == IDLE) {
        TKComplete( rtn_code, errtxt, -1 );
    }
    return TK_CONTINUE;
}

int dimread( int chan )
{
    char *par;
    char errtxt[LBUFF_SIZE+1];
    int rtn_code;

    TKAck( 5, -1);

    if(!TKGetString("par",&par)) { /* error return */
        sprintf(stderr,"Error in Read,Parameter code:empty.\n");
        return TK_CONTINUE;
    }
    if(read_state(stateP, par, errtxt) != 0) {
        sprintf(stderr,"Error in Read,Parameter code:%s.\n",par);
        sprintf(errtxt,"Error in Read,Parameter code:%s",par);
        rtn_code = -1;
    } else {
        rtn_code = 0;
    }

    TKComplete( rtn_code, errtxt, -1 );
    return TK_CONTINUE;
}

int idle( long runtime )
{
    char buffer[BUFSIZ+1];
    Rtn_msg_buf rmsgbuf; /* message send structure */
    Rtn_msg_bufP rmsgp;
    long msotyp;
    FILE *res_fid;
    float recovery;
    char res_fname[LBUFF_SIZE];
    char errtxt[LBUFF_SIZE+1];

    rmsgp = &rmsgbuf;
}

```

```

if( getstdin( buffer ) == 1)
    switch( processkeys( buffer ) )
    {
        case SEND_COMMAND:
            TKsendMessage( buffer );
            break;
        case SEND_DATA:
            TKsendData( (void *)datatosend, strlen(datatosend) + 1 );
            break;
        case QUIT_COMMAND:
            return TK_STOP;
            break;
        case SEND_QUERY:
            TKQuery("What do I do now?", "eat work", -1 );
            break;
    }

switch( slmstate ) {

case ADDSTD:
    break;
case VALIDATE:
/* receive message from matlab engine process */
    msgtyp = 0;
    if(msgrecv(rmsqid, rmsgp, MSG_SIZE, msgtyp, IPC_NOWAIT)>=0) {
        printf("\nresponse from peer received\n");
        slmstate = IDLE;
/* set appropriate flags */
        if(rmsgp->code == 0) {
            fprintf(stderr, "Matlab call successful\n");
            if(strcasecmp(stateP->samp_type, "control") == 0) {
                stateP->qa_cal_flag = 1;
            } else {
                sscanf(rmsgp->result,"%g",&recovery);
                stateP->surr_recover = recovery;
                stateP->qa_sur_flag = 1;
            }
            TKComplete( rmsgp->code, "Validate complete", -1 );
        } else {
            strcat(errtxt, decipher_err_codes(rmsgp->code,1));
            strcat(errtxt, decipher_err_codes(rmsgp->code,2));
            fprintf(stderr, "%s.\n",errtxt);
            fprintf(stderr, "Matlab call error\n");
            if(strcasecmp(stateP->samp_type, "control") == 0) {
                stateP->qa_cal_flag = -1;
            } else {
                sscanf(rmsgp->result,"%g",&recovery);
                stateP->surr_recover = recovery;
                stateP->qa_sur_flag = -1;
            }
        }
    }
}

```

```

        }
        TKComplete( rmsgp->code, errtxt, -1 );
    }

/* write results to ASCII results file */
strcpy(res_fname,stateP->batch_path);
strcat(res_fname,"/");
strcat(res_fname, RESULTS_FILE);
sprintf(stderr,"results file: %s\n",res_fname);
if((res_fid = fopen(res_fname, "a+")) == (FILE *)NULL) {
    sprintf(stderr,"Could not open %s to append results\n",res_fname);
    sprintf(stderr,"Exiting !!\n");
    return TK_STOP;
}
fseek(res_fid, 0L, SEEK_END);
fprintf(res_fid,"%s ",stateP->samp_id);
fprintf(res_fid,"%s ",stateP->samp_type);
fprintf(res_fid,"%-10.3e ",stateP->samp_amt);
fprintf(res_fid,"%s ",stateP->datafile);
fprintf(res_fid,"%s ",stateP->calib_set);
fprintf(res_fid,"%- 3i ",stateP->qa_scale_flag);
fprintf(res_fid,"%- 3i ",stateP->qa_rt_flag);
fprintf(res_fid,"%- 3i ",stateP->qa_cal_flag);
fprintf(res_fid,"%- 3i ",stateP->comb_mode);
fprintf(res_fid,"%-10.3e ",stateP->surr_amt);
fprintf(res_fid,"%-10.3e ",stateP->surr_recover);
fprintf(res_fid, "\n");
/* insert newline for all but type unknown or if qc check failed*/
/*
if (strcasecmp(stateP->samp_type, "unknown") != 0 ||
    (rmsgp->code == 0)) {
    fprintf(res_fid, "\n");
}
*/
fclose(res_fid);
}
break;
case ANALYZE:
msgtyp = 0;
if(msgrcv(rmsqid, rmsgp, MSG_SIZE, msgtyp, IPC_NOWAIT)>=0) {
printf("\nresponse from peer received\n");
slmstate = IDLE;
if (rmsgp->code == 0) {
    TKComplete( rmsgp->code, rmsgp->result, -1 );
    sprintf(stderr, "Matlab call successful\n");
} else {
    sprintf(stderr, "Matlab call error\n");
    strcat(errtxt, decipher_err_codes(rmsgp->code,1));
    strcat(errtxt, decipher_err_codes(rmsgp->code,2));
    sprintf(stderr,"%s.\n",errtxt);
}
}

```

```

        TKComplete( rmsgp->code, errtxt, -1);
    }
    strcpy(res_fname,stateP->batch_path);
    strcat(res_fname,"/");
    strcat(res_fname, RESULTS_FILE);
    fprintf(stderr,"results file: %s\n",res_fname);
    if((res_fid = fopen(res_fname, "a+")) == (FILE *)NULL) {
        fprintf(stderr,"Could not open %s to append results\n",res_fname);
        fprintf(stderr,"Exiting !!\n");
        return TK_STOP;
    }
    fseek(res_fid, 0L, SEEK_END);
    fseek(res_fid, -1L, SEEK_CUR); /*remove newline from validate*/
    fprintf(res_fid,"%s\n",rmsgp->result);
    fclose(res_fid);
}
/* reset state */
sprintf(stateP->samp_type, DEF_SAMP_TYPE);
sprintf(stateP->samp_id, DEF_SAMP_ID);
sprintf(stateP->datafile, DEF_DATAFILE);
stateP->surr_recover = DEF_SUR_RECOVER;
stateP->samp_amt = DEF_SAMP_AMT;
stateP->qa_scale_flag = DEF_QA_SCALE;
stateP->qa_rt_flag = DEF_QA_RT;
stateP->qa_cal_flag = DEF_QA_CAL;
stateP->qa_sur_flag = DEF_QA_SUR;

break;
case SET_BATCH:
    msgtyp = 0;
    if(msgrcv(rmsqid, rmsgp, MSG_SIZE, msgtyp, IPC_NOWAIT)>=0) {
        printf("\nresponse from peer received\n");
        slmstate = IDLE;
        if(rmsgp->code == 0) {
            fprintf(stderr, "Matlab call successful\n");
            TKComplete( rmsgp->code, "Set Batch Path complete", -1 );
        } else {
            fprintf(stderr, "Matlab call error\n");
            strcat(errtxt, decipher_err_codes(rmsgp->code,1));
            strcat(errtxt, decipher_err_codes(rmsgp->code,2));
            fprintf(stderr,"%s.\n",errtxt);
            TKComplete( rmsgp->code, errtxt, -1 );
        }
    }
break;
case IDLE:
    msgtyp = 0;

```

```

        if(msgrcv(rmsqid, rmsgp, MSG_SIZE, msgtyp, IPC_NOWAIT)>=0) {
            printf("\nresponse from peer received\n");
            if (rmsgp->code == 0) {
                fprintf(stderr, "Matlab call successful\n");
            } else {
                fprintf(stderr, "Matlab call error\n");
            }
        }

        break;
    }
    return TK_CONTINUE;
}

int bypass( char *channel, char *cmdstring )
{
    char errtxt[LBUFF_SIZE+1];
    int rtn_code=0;
    Cmd_msg_buf tmsgbuf; /* message send structure */
    Cmd_msg_bufP rmsgp;
    int msgsz, msgflg;
    long msgtyp;
    struct msqid_ds tmsgds;
    struct msqid_ds *tmsgdsp;
    struct msqid_ds rmsqds;
    struct msqid_ds *rmmsgdsp;
    struct shmid_ds shmds;
    struct shmid_ds *shmdsp;

    rmsgp = &tmsgbuf;
    tmsgdsp = &tmsgds;
    rmmsgdsp = &rmsgds;
    shmdsp = &shmds;

    rmsgp->mtype = 1;
    msgflg = 0;

    printf("bypass- %s %s\n", channel, cmdstring );
    strncpy(rmsgp->command, cmdstring, LBUFF_SIZE-2);

    msgsز = strlen(cmdstring) + 1;

    /*
     * send message to queue
     */
    rtn_code = msgsnd(tmsqid, rmsgp, msgsز, msgflg);
    if (rtn_code == -1) {

```

```

    perror("msgop: msgsnd failed");
    sprintf(errtxt, "Message send failed, in Bypass");
}

sprintf(errtxt, "Message sent to process running MATLAB engine");

msgctl(tmsqid, IPC_STAT, tmsgdsp);
msgctl(rmsqid, IPC_STAT, rmsgdsp);
shmctl(shmid, IPC_STAT, shmdsp);

fprintf(stderr,"number of messages on transmit que:%d\n",tmsgdsp->msg_qnum);
fprintf(stderr,"number of messages on receive que:%d\n",rmsgdsp->msg_qnum);
fprintf(stderr,"number of shared memory attachments:%d\n",shmdsp->shm_nattch);

slmstate = IDLE;

TKIAck(rtn_code, errtxt );

return TK_CONTINUE;
}

int intraLUO( int cmd, int channel )
{
    char responsetxt[128];
    char *response;

    printf("intraLUO - %d %d\n", cmd, channel);
    switch( cmd ){
        case ABORT_CMD:
            TKsend( 1, "%dRTN", 0, "KEY" , "ABORT", "TXT", "Aborting DIM processing", 0 );
            slmstate = IDLE;
            break;
        case REPLY_CMD:
            if( TKGetString( "txt", &response ) )
                printf("Response = %s\n", response );
            TKIAck( 0, NULL );
            break;
        case STATUS_CMD:
            if( slmstate == IDLE ) {
                TKIAck( 0, "IDLE" );
                print_state(stateP);
            } else {
                sprintf( responsetxt, "%s", TKgetLastCommand() );
                TKIAck( 0, responsetxt );
            }
            break;
    }
}

```

```

        }

    return TK_CONTINUE;
}

int error( int error )
{
    return TK_ABORT;
}

int comStatus(int type, char *message)
{
    switch(type) {
        case SERVER_CONNECTED:
            printf("SERVER CONNECTED ");
            break;
        case CONNECTION_DISCONNECTED:
            printf("CONNECTION DISCONNECTED ");
            break;
        case BROADCAST_MESSAGE:
            printf("BROADCAST MESSAGE ");
            break;
    }
    printf(message);
    printf("\r\n");
    return TK_CONTINUE;
}

main( argc, argv )
int argc;
char *argv[];
{
    char *myname;           /* pointer to name of program */
    char *portnum;          /* pointer to remote user name */
    int theServer = -1;
    int len, ret;
    struct TKcmd *cmdtab = slmCmds;
    char matlab_strt[LBUFF_SIZE];
    int mat_out_size = MATLAB_OUTPUT_SIZE;

    key_t      tkey;      /* unique key for send message queue */
    key_t      rkey;      /* unique key for receive message queue */
    int       cld_pid; /* child pid */
}

myname = argv[0];

```

```

/* Check that there is one command line argument */

if( (argc < 4 ) || ( argc > 5 ) ) {
    fprintf( stderr, "Usage: %s slmid service broadcastport \n", myname );
    fprintf( stderr, "OR\n");
    fprintf( stderr, "Usage: %s slmid service host port\n", myname );
    exit(1);
}

/* set the umask mode */
umask(07); /* no other permission, user and group as requested */

switch ((cld_pid=fork())) {
case -1:           /* error */
    perror("main:fork");
    exit(1);
case 0:            /* child */
    execl("matlab_eng", "matlab_eng", (char *)NULL);
    perror("main:execl");
    exit(1);
case 1:           /* parent */
    break;
}

/* set up message queues and shared memory */

/* */
/* generate keys based on file and code          */
/* */
tkey = ftok("/usr", 'A');
rkey = ftok("/usr", 'B');

if(rkey == -1 || tkey == -1) {
    fprintf(stderr,"ftok failure in %s, exiting\n",argv[0]);
    exit(1);
}

/* */
/* create the message channel          */
/* */

if ((tmsqid = msgget(tkey, (IPC_CREAT | 0666))) == -1) {
    perror("msgget: msgget failed");
    exit(1);
}

```

```

if ((rmsqid = msgget(rkey, (IPC_CREAT | 0666))) == -1) {
    perror("msgget: msgget failed");
    exit(1);
}

/* */
/* create shared memory segment */
/* */

if ((shmid = shmget(tkey, 600000, (IPC_CREAT | 0666))) == -
1) {
    perror("shmget: shmget failed");
    exit(1);
}
}

stateP = &state;

TKSetSLMInfo( "dim", argv[1], "ORNL", 1, "Ver. 1.0" );

if( argc == 4 )
    TKSetConBroadcast( argv[2], atoi( argv[3] ) );
else
    TKSetConDirect( argv[2], argv[3], atoi( argv[4] ) );

init_state(stateP);

ret = TKexecutive( &slmCmds[0], idle, bypass, intraLUO, error, comStatus, 0 );

/* remove message queues and shared memory */
msgctl(rmsqid, IPC_RMID, (struct msqid_ds *) NULL);
msgctl(shmid, IPC_RMID, (struct shmid_ds *) NULL);
shmctl(shmid, IPC_RMID, (struct shmid_ds *) NULL);

/* kill child process */
if(kill(cld_pid, SIGKILL))
    perror("kill: kill child failed");

switch (ret ){
case TK_ABORT:
    printf("Error in TKExec - %s\n", TKgetLastError());
    printf("Aborting\n");
    exit(1);
    break;
case TK_STOP:

```

```

        printf("Stopping\n");
        exit(0);
        break;
    }

}

dim_slm.h
#ifndef __DIM_SLM__
#define __DIM_SLM__

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
/*#include "cmapi.h"*/
#include "engine.h"

#define LBUFF_SIZE 490
#define MSG_SIZE 500
#define RESULTS_FILE "results.txt"

typedef struct dim_state {
    int comb_mode;      /* results combination mode */
    char batch_path[LBUFF_SIZE]; /* complete pathname to base level of batch */
    char samp_type[LBUFF_SIZE];   /* sample type AIA header */
    char samp_id[LBUFF_SIZE];    /* sample id AIA header */
    char datafile[LBUFF_SIZE];   /*full base pathname of raw chromatogram */
                           /* and ASCII peak area file */
    float inj_vol;          /* injection volume in uL*/
    float samp_amt;         /* sample amount in mg */
    float surr_amt;         /* surrogate amount in ng */
    float surr_recover;     /* surrogate recovery in percent */
    char calib_set[LBUFF_SIZE]; /* prefix of the filename of the cal set */
    char surr_set[LBUFF_SIZE]; /* prefix of the filename of the surrogate set*/
    int qa_scale_flag;      /* results of qa off-scale signal check*/
    int qa_rt_flag;         /* results of qa retention time check */
    int qa_sur_flag;        /* results of surrogate recovery check */
    int qa_cal_flag;        /* results of qa calibration check */
} Dim_state;

typedef struct dim_state *Dim_stateP;

typedef struct cmd_msg_buf {
    long mtype;           /* message type */
    char command[LBUFF_SIZE]; /*MATLAB command string */

```

```

} Cmd_msg_buf;

typedef struct cmd_msg_buf *Cmd_msg_bufP;

typedef struct rtn_msg_buf {
    long mtype;           /* message type */
    int code;             /* variable code from matlab environemnt */
    char result[LBUFF_SIZE]; /* MATLAB command string */
} Rtn_msg_buf;

typedef struct rtn_msg_buf *Rtn_msg_bufP;

int getstdin( char *buffer );
int processkeyin( char *buffer);
int processcmds( char *buf );
int init_state(Dim_stateP stateP);
int set_state(Dim_stateP stateP, char *paramcptr, char *valcptr);
int read_state(Dim_stateP stateP, char *paramcptr, char *valcptr);
int print_state(Dim_stateP stateP);
void iluo_action();
int aia_poke(const char *cdfname, char *type_val, float amt_val);
void dim_err_code(int err_no, char *buffer);
int check_state(Dim_stateP stateP);
int validate(Dim_stateP stateP);
char *decipher_err_codes(int, int);
extern void      exit();
extern void      perror();
extern char   *malloc();
extern char   *shmat();

#define MATLAB_OUTPUT_SIZE 20000

enum {
    NO_COMMAND = 0,
    SEND_COMMAND,
    SEND_DATA,
    SEND_QUERY,
    QUIT_COMMAND };

enum {
    IDLE = 0,
    ADDSTD,
    VALIDATE,
    ANALYZE,
    SET_BATCH};

```

```

enum {
    IDENTIFY = 0,
    INITIALIZE,
    START_VAL,
    START_ANA,
    SET,
    READ,
    ABORT,
    STATUS,
    REPLY,
    BYPASS,
    INVALID
};

enum {
    COMB_MIN = 0,
    COMB_MAX,
    COMB_AVG,
    COMB_FUS,
    PCRR,
    PCRP,
    MLRR,
    MLRP,
    LRP,
    NNP,
    SLER
};

enum {
    INIT_ERR = -200,
    BAD_ARG,
    MATLAB_ENGINE_OPEN,
    BATCH_DIR_UNSET,
    SAMPLE_ID_UNSET,
    DATAFILE_UNSET,
    SAMPLE_AMT_UNSET,
    SURROGATE_AMT_UNSET,
    SURROGATE_NAME_UNSET
};

#define DEF_COMB_MODE COMB_MAX
#define DEF_BATCH_PATH "\0"
#define DEF_SAMP_TYPE "unknown"
#define DEF_SAMP_ID "\0"
#define DEF_DATAFILE "\0"
#define DEF_INJ_VOL 0.2
#define DEF_SAMP_AMT -0.1

```

```

#define DEF_SUR_AMT -0.1
#define DEF_SUR_RECOVER 0.0
#define DEF_CALIB_SET "\0"
#define DEF_SUR_SET "\0"
#define DEF_QA_SCALE -1
#define DEF_QA_RT -1
#define DEF_QA_CAL 0
#define DEF_QA_SUR -1

#endif

```

### **dim\_util.c**

```

#include "dim_slm.h"
#include <sys/ipc.h>

```

```

/* parse commands typed into the stdin of the dim_slm program */
int processkeys( char *buf )
{
    int len;
    char *cptr, *tptr, *remain;

    if( strlen( buf ) == 1 ) {
        switch( buf[0] ) {
            case '\n':
            case '?':
                break;
        }
        return( NO_COMMAND );
    }

    if( strcmp( "Quit\n", buf ) == 0 )
        return( QUIT_COMMAND );

    if( strcmp( "Send\n", buf ) == 0 )
        return( SEND_DATA );

    if( strcmp( "Query\n", buf ) == 0 )
        return( SEND_QUERY );

    return(SEND_COMMAND);
}

int processcmds( char *buf)

```

```

{
    char *cptr;
    int command;

    printf( "COMMAND_MESSAGE - %s\n", buf);
    if( (cptr = strchr( buf, '\n')) != NULL ) {
        *cptr = '\0';
    }
    if( (cptr = strtok( buf, " " )) != NULL ) {
        if(strcasecmp(cptr, "Identify") == 0) { /* Identify cmd */
            command = IDENTIFY;
        } else if(strcasecmp(cptr, "Initialize") == 0) {
            command = INITIALIZE;
        } else if(strcasecmp(cptr, "Start") == 0) {
            cptr = strtok(NULL, " ");
            if(strcasecmp(cptr,"Validate") == 0) {
                command = START_VAL;
            } else if(strcasecmp(cptr,"Analyze") == 0) {
                command = START_ANA;
            } else {
                command = INVALID;
            }
        } else if(strcasecmp(cptr, "Set") == 0) {
            command = SET;
        } else if(strcasecmp(cptr, "Read") == 0) {
            command = READ;
        } else if(strcasecmp(cptr, "Abort") == 0) {
            command = ABORT;
        } else if(strcasecmp(cptr, "Status") == 0) {
            command = STATUS;
        } else if(strcasecmp(cptr, "Reply") == 0) {
            command = REPLY;
        } else if(strcasecmp(cptr, "Bypass") == 0) {
            command = BYPASS;
        } else {
            command = INVALID;
        }
    } else {
        command = INVALID;
    }

    return (command);
}

int init_state(Dim_stateP stateP)
{
    stateP->comb_mode = DEF_COMB_MODE;
}

```

```

sprintf(stateP->batch_path, DEF_BATCH_PATH);
sprintf(stateP->samp_type, DEF_SAMP_TYPE);
sprintf(stateP->samp_id, DEF_SAMP_ID);
sprintf(stateP->datafile, DEF_DATAFILE);
stateP->inj_vol = DEF_INJ_VOL;
stateP->samp_amt = DEF_SAMP_AMT;
stateP->surr_recover = DEF_SUR_RECOVER;
sprintf(stateP->calib_set, DEF_CALIB_SET);
sprintf(stateP->surr_set, DEF_CALIB_SET);
stateP->qa_scale_flag = DEF_QA_SCALE;
stateP->qa_rt_flag = DEF_QA_RT;
stateP->qa_sur_flag = DEF_QA_SUR;
stateP->qa_cal_flag = DEF_QA_CAL;
return(0);
}

int print_state(Dim_stateP stateP)
{
    switch(stateP->comb_mode){
        case COMB_MIN:
            fprintf(stderr,"Results combination mode: Minimum. \n");
            break;
        case COMB_MAX:
            fprintf(stderr,"Results combination mode: Maximum. \n");
            break;
        case COMB_AVG:
            fprintf(stderr,"Results combination mode: Average. \n");
            break;
        case COMB_FUS:
            fprintf(stderr,"Results combination mode: Fusion. \n");
            break;
        case PCRR:
            fprintf(stderr,"Results combination mode: PCR Raw. \n");
            break;
        case PCRP:
            fprintf(stderr,"Results combination mode: PCR Peak. \n");
            break;
        case MLRR:
            fprintf(stderr,"Results combination mode: MLR Raw. \n");
            break;
        case MLRP:
            fprintf(stderr,"Results combination mode: MLR Peak. \n");
            break;
        case LRP:
            fprintf(stderr,"Results combination mode: LR Peak. \n");
            break;
        case NNP:
            fprintf(stderr,"Results combination mode: Neural Network Peak. \n");
    }
}

```

```

        break;
    case SLER:
        fprintf(stderr,"Results combination mode: SLE Raw. \n");
        break;
    default:
        fprintf(stderr,"Unknown combination mode - error! \n");
        return(-1);
    }
    fprintf(stderr,"Batch directory path name: %s \n",stateP->batch_path);
    fprintf(stderr,"Sample type: %s \n",stateP->samp_type);
    fprintf(stderr,"Sample ID: %s \n",stateP->samp_id);
    fprintf(stderr,"Datafile name: %s \n",stateP->datafile);
    fprintf(stderr,"Injection volume: %f uL.\n", stateP->inj_vol);
    fprintf(stderr,"Sample amount: %f mg.\n", stateP->samp_amt);
    fprintf(stderr,"Surrogate amount: %f ng.\n", stateP->surr_amt);
    fprintf(stderr,"Surrogate recovery: %f percent.\n", stateP->surr_recover);
    fprintf(stderr,"Calibration set prefix: %s\n",stateP->calib_set);
    fprintf(stderr,"Surrogate set prefix: %s\n",stateP->surr_set);
    fprintf(stderr,"QA off-scale flag: %d\n", stateP->qa_scale_flag);
    fprintf(stderr,"QA retention time flag: %d\n", stateP->qa_rt_flag);
    fprintf(stderr,"QA surrogate check flag: %d\n", stateP->qa_sur_flag);
    fprintf(stderr,"QA calibration check flag: %d\n", stateP->qa_cal_flag);
    return(0);
}

int set_state(Dim_stateP stateP, char *paramcptr, char *valcptr)
{
    int flag=0;
    int ret, rtn_code;
    float vol;
    float amt;
    static char file_name[LBUFF_SIZE];
    char *tcptr;
    char cmdstring[LBUFF_SIZE];
    FILE *fp;

    Cmd_msg_buf tmsgbuf; /* message send structure */
    Cmd_msg_bufP tmsgp;
    int msgsiz, msgflg;
    long msqid;

    extern int slmstate;
    extern int tmsqid;

    tmsgp = &tmsgbuf;

    tmsgp->mtype = 1;
    msgflg = 0;
}

```

```

fprintf(stderr,"set_state parameter:%s  value:%s \n",paramcptr,valcptr);

if (*paramcptr==(char *)NULL) {
    fprintf(stderr,"Null value for parameter, exiting\n");
    flag = -1;
    return(flag);
}
if (*valcptr==(char *)NULL) {
    fprintf(stderr,"Null value for value, exiting\n");
    flag = -1;
    return(flag);
}

if((tcptr = strchr(paramcptr,'\'\n'))!= (char *)NULL) {
    *tcptr = '\0';
}
if((tcptr = strchr(valcptr,'\'\n'))!= (char *)NULL) {
    *tcptr = '\0';
}

if (strcasecmp(paramcptr, "comb") == 0) { /* Comb. mode */
    if (strcasecmp(valcptr, "Max") == 0) {
        stateP->comb_mode = COMB_MAX;
    } else if (strcasecmp(valcptr, "Min") == 0) {
        stateP->comb_mode = COMB_MIN;
    } else if (strcasecmp(valcptr, "Avg") == 0) {
        stateP->comb_mode = COMB_AVG;
    } else if (strcasecmp(valcptr, "Fusion") == 0) {
        stateP->comb_mode = COMB_FUS;
    } else if (strcasecmp(valcptr, "pcrr") == 0) {
        stateP->comb_mode = PCRR;
    } else if (strcasecmp(valcptr, "pcrp") == 0) {
        stateP->comb_mode = PCRP;
    } else if (strcasecmp(valcptr, "mlrr") == 0) {
        stateP->comb_mode = MLRR;
    } else if (strcasecmp(valcptr, "mlrp") == 0) {
        stateP->comb_mode = MLRP;
    } else if (strcasecmp(valcptr, "lrp") == 0) {
        stateP->comb_mode = LRP;
    } else if (strcasecmp(valcptr, "nnp") == 0) {
        stateP->comb_mode = NNP;
    } else if (strcasecmp(valcptr, "sler") == 0) {
        stateP->comb_mode = SLER;
    } else {
        flag = -1; /* no change, invalid mode */
    }
}

```

```

slmstate = IDLE;
} else if(strcasecmp(paramcptr, "batch_path") == 0) {

    sprintf(stateP->batch_path, valcptr);
    if(strcmp(stateP->batch_path, "\\0")!=0) { /* batch path has been set */
/* code to run matlab function follows */

    sprintf(cmdstring,"code = dim_set_batch('%s')",stateP->batch_path);
    fprintf(stderr,"Matlab command string:%s\n",cmdstring);
    strncpy(tmmsgp->command, cmdstring, LBUFF_SIZE-2);

    msgsz = strlen(cmdstring) + 1;

/* */
/* send message to queue */
/* */
rtn_code = msgsnd(tmqid, tmmsgp, msgsz, msgflg);
if (rtn_code == -1) {
    perror("msgop: msgsnd failed");
}

} else {
    flag = -1;
    fprintf(stderr,"In set batch path: null path argument\n");
}

slmstate = SET_BATCH;
} else if(strcasecmp(paramcptr, "AIA_samp_type") == 0) {
if ((strcasecmp(valcptr, "standard") == 0) ||
    (strcasecmp(valcptr, "unknown") == 0) ||
    (strcasecmp(valcptr, "control") == 0) ||
    (strcasecmp(valcptr, "blank") == 0)) {
    sprintf(stateP->samp_type, valcptr);
} else {
    flag = -1;
}

slmstate = IDLE;
} else if(strcasecmp(paramcptr, "AIA_samp_id") == 0) {
    sprintf(stateP->samp_id, valcptr);

    slmstate = IDLE;
} else if(strcasecmp(paramcptr, "datafile_name") == 0) {
/* check to see if batch_path and samp_id are set */
    if (strcmp(stateP->batch_path, DEF_BATCH_PATH) &&
        strcmp(stateP->samp_id, DEF_SAMP_ID) &&
        !(stateP->samp_amt == DEF_SAMP_AMT)) {

```

```

sprintf(stateP->datafile, valcptr);
/* check to see file exists*/
if((fp=fopen(valcptr, "r"))==(FILE *)NULL) {
    flag = -1;
    fprintf(stderr,"could not open %s for reading.\n",valcptr);
} else {
    fclose(fp);

/* create directory */
/* check to see if directory/file exists */
sprintf(file_name,"%s/%ss/%s.d",
        stateP->batch_path, stateP->samp_type, stateP->samp_id);

if((fp=fopen(file_name, "r"))!=(FILE *)NULL) {
    fprintf(stderr,"%s already exists, no copy, using existing file\n",file_name);
    fclose(fp);
    flag = 1;
    ret = 0;
} else {
    sprintf(cmdstring,"mkdir %s/%ss/%s.d",
            stateP->batch_path, stateP->samp_type, stateP->samp_id);
    ret = system(cmdstring);
}

if(ret != 0) {
    flag = -1;
} else { /* mkdir successful, copy the file */
    if(flag != 1) {
        sprintf(cmdstring,"chmod 2770 %s/%ss/%s.d",
                stateP->batch_path, stateP->samp_type, stateP->samp_id);
        if(system(cmdstring) != 0) {
            flag = -1;
        } else { /* chmod successful, copy the file */

/* copy the file */
        sprintf(file_name,"%s/%ss/%s.d/%s.cdf",
                stateP->batch_path, stateP->samp_type,
                stateP->samp_id, stateP->samp_id);
        sprintf(cmdstring,"cp %s %s",valcptr, file_name);

        if(system(cmdstring) != 0) {
            flag = -1;
        } else { /* copy successful, poke AIA values */
            amt = stateP->samp_amt;
            if(aia_poke(file_name, stateP->samp_type, amt) == 0) {
                fprintf(stderr,"poke AIA header values\n");
            } else {
                flag = -1;
                fprintf(stderr,"could not poke AIA header values in %s\n"

```

```

                ,file_name);
            }
        }
    }
    flag = 0; /* work around for warning causing an Error state in TSC*/
}
}
} else {
    flag = -1;
    sprintf(stderr,"Batch path, sample ID, or sample amt not set.\n");
}

slmstate = IDLE;
} else if(strcasecmp(paramcptr, "AIA_inj_vol") == 0) {
    if(sscanf(valcptr,"%f", &vol) > 0) {
        stateP->inj_vol = vol;
    } else {
        sprintf(stderr,"Could not convert %s to injection column\n",valcptr);
        flag = -1; /* not proper format for numeric conversion */
    }
    slmstate = IDLE;
} else if(strcasecmp(paramcptr, "AIA_samp_amt") == 0) {
    if((ret=sscanf(valcptr,"%f %s", &amt, cmdstring)) > 0) {
/* check for units */
        if(ret == 2) { /* units present */
            if(strcasecmp(cmdstring,"mg") == 0) {
                stateP->samp_amt = amt;
            }else if (strcasecmp(cmdstring,"g") == 0) {
                amt = amt * 1000.0;
                stateP->samp_amt = amt;
            }else if (strcasecmp(cmdstring,"kg") == 0) {
                amt = amt * 1000000.0;
                stateP->samp_amt = amt;
            }else {
                sprintf(stderr,"Could not convert %s to MKS units.\n",cmdstring);
                flag = -1; /* undefined units */
            }
        } else { /* assume units in g */
            stateP->samp_amt = amt * 1000.0;
        }
    } else {
        sprintf(stderr,"Could not convert %s to sample amount.\n",valcptr);
        flag = -1; /* not proper format for numeric conversion */
    }
    slmstate = IDLE;
} else if(strcasecmp(paramcptr, "surr_amt") == 0) {
    if((ret=sscanf(valcptr,"%f %s", &amt, cmdstring)) > 0) {
/* check for units */

```

```

        if(ret == 2) { /* units present */
            if(strcasecmp(cmdstring,"ng") == 0) {
                stateP->surr_amt = amt;
            } else if (strcasecmp(cmdstring,"mg") == 0) {
                amt = amt * 1000.0;
                stateP->surr_amt = amt;
            } else if (strcasecmp(cmdstring,"g") == 0) {
                amt = amt * 1000000.0;
                stateP->surr_amt = amt;
            } else {
                fprintf(stderr,"Could not convert %s to MKS units.\n",cmdstring);
                flag = -1; /* undefined units */
            }
        } else { /* assume units in ng */
            stateP->surr_amt = amt;
        }
    } else {
        sprintf(stderr,"Could not convert %s to surrogate amount.\n",valcptr);
        flag = -1; /* not proper format for numeric conversion */
    }
    slmstate = IDLE;
} else if(strcasecmp(paramcptr, "surr_name") == 0) {
    sprintf(stateP->surr_set, valcptr);

    slmstate = IDLE;
} else {
    fprintf(stderr,"Invalid parameter to set: %s.\n",paramcptr);
    flag = -1; /* invalid parameter */
    slmstate = IDLE;
}

return(flag);
}

```

```

int read_state(Dim_stateP stateP, char *paramcptr, char *valcptr)
{
    int flag=0;
    float vol;
    char *tcptr;

    if((tcptr = strchr(paramcptr,'\'\n'))!= (char *)NULL) {
        *tcptr = '\0';
    }
    if (strcasecmp(paramcptr, "comb") == 0) { /* Comb. mode */
        switch(stateP->comb_mode){
        case COMB_MIN:
            sprintf(stderr,"Results combination mode: Minimum. \n");
            sprintf(valcptr,"MIN\0");

```

```

        break;
    case COMB_MAX:
        fprintf(stderr,"Results combination mode: Maximum. \n");
        sprintf(valcptr,"MAX\0");
        break;
    case COMB_AVG:
        fprintf(stderr,"Results combination mode: Average. \n");
        sprintf(valcptr,"AVG\0");
        break;
    case COMB_FUS:
        fprintf(stderr,"Results combination mode: Fusion. \n");
        sprintf(valcptr,"FUSION\0");
        break;
    default:
        fprintf(stderr,"Unknown combination mode - error! \n");
        return(-1);
    }
} else if(strcasecmp(paramcptr, "Batch_path") == 0) {
    fprintf(stderr,"Batch_path:%s\n",stateP->batch_path);
    sprintf(valcptr,"%s\0",stateP->batch_path);
} else if(strcasecmp(paramcptr, "AIA_samp_type") == 0) {
    fprintf(stderr,"AIA_sample_type:%s\n",stateP->samp_type);
    sprintf(valcptr,"%s\0",stateP->samp_type);
} else if(strcasecmp(paramcptr, "AIA_samp_id") == 0) {
    fprintf(stderr,"AIA_sample_id:%s\n",stateP->samp_id);
    sprintf(valcptr,"%s\0",stateP->samp_id);
} else if(strcasecmp(paramcptr, "datafile_name") == 0) {
    fprintf(stderr,"Datafile_name:%s\n",stateP->datafile);
    sprintf(valcptr,"%s\0",stateP->datafile);
} else if(strcasecmp(paramcptr, "AIA_inj_vol") == 0) {
    fprintf(stderr,"AIA_inj_vol:%f\n",stateP->inj_vol);
    sprintf(valcptr,"%f\0",stateP->inj_vol);
} else if(strcasecmp(paramcptr, "AIA_samp_amt") == 0) {
    fprintf(stderr,"AIA_samp_amt:%f\n",stateP->samp_amt);
    sprintf(valcptr,"%f\0",stateP->samp_amt);
} else if(strcasecmp(paramcptr, "surr_amt") == 0) {
    fprintf(stderr,"Surrogate amount:%f ng\n",stateP->surr_amt);
    sprintf(valcptr,"%f\0",stateP->surr_amt);
} else if(strcasecmp(paramcptr, "surr_recover") == 0) {
    fprintf(stderr,"Surrogate recovery:%f percent\n",stateP->surr_recover);
    sprintf(valcptr,"%f\0",stateP->surr_amt);
} else if(strcasecmp(paramcptr, "cal_prefix") == 0) {
    fprintf(stderr,"Calibration file prefix:%s\n",stateP->calib_set);
    sprintf(valcptr,"%s\0",stateP->calib_set);
} else if(strcasecmp(paramcptr, "surr_name") == 0) {
    fprintf(stderr,"Surrogate name:%s\n",stateP->surr_set);
    sprintf(valcptr,"%s\0",stateP->surr_set);
} else {
    flag = -1; /* invalid parameter */
}

```

```

        }
        return(flag);
    }

void dim_err_code(int err_no, char *buffer)
{
    switch(err_no) {
        case INIT_ERR:
            strcpy(buffer,"Initialization failed");
            break;
        case BAD_ARG:
            strcpy(buffer,"Bad Argument for command");
            break;
        case MATLAB_ENGINE_OPEN:
            strcpy(buffer,"Could not open MATLAB engine");
            break;
        case BATCH_DIR_UNSET:
            strcpy(buffer,"Pathname to batch processing directory not set");
            break;
        case SAMPLE_ID_UNSET:
            strcpy(buffer,"Sample ID not set");
            break;
        case DATAFILE_UNSET:
            strcpy(buffer,"Filename of raw AIA file not set");
            break;
        default:
            if (err_no < 0) {
                sprintf(buffer,"%s %s",decipher_err_codes(err_no,1),
                        decipher_err_codes(err_no,2));
            } else {
                strcpy(buffer,"No error text");
            }
            break;
    }
}

/* check to see if necessary parameters have been set */

int check_state(Dim_stateP stateP)
{
    int flag;

    flag = 0;

    if (!strcmp(stateP->batch_path, DEF_BATCH_PATH)) {
        flag = BATCH_DIR_UNSET;
    } else if (!strcmp(stateP->samp_id, DEF_SAMP_ID)) {
        flag = SAMPLE_ID_UNSET;
    }
}

```

```

} else if (!strcmp(stateP->datafile, DEF_DATAFILE)) {
    flag = DATAFILE_UNSET;
} else if (stateP->samp_amt == DEF_SAMP_AMT) {
    flag = SAMPLE_AMT_UNSET;
} else if (stateP->surr_amt == DEF_SUR_AMT) {
    flag = SURROGATE_AMT_UNSET;
} else if (stateP->surr_set == DEF_SUR_SET) {
    flag = SURROGATE_NAME_UNSET;
}

return(flag);
}

/* validate gc */

int validate(Dim_stateP stateP)
{
    char      cmdstring[LBUFF_SIZE+1];
    int       rtn, rtn_code;
    float     recovery;
    Rtn_msg_buf rmsgbuf; /* message send structure */
    Rtn_msg_bufP rmsgp;
    long      msgtyp;
    Cmd_msg_buf tmsgbuf; /* message send structure */
    Cmd_msg_bufP tmsgp;
    int       msgsiz, msgflg;
    extern int tmsqid, rmsqid;
    extern int slmstate;

    tmsgp = &tmsgbuf;
    rmsgp = &rmsgbuf;

    tmsgp->mtype = 1;
    msgflg = 0;

    /* get calibration file prefix */
    sprintf(cmdstring,[result, code] = cal_prefix;");
    fprintf(stderr,"%s\n", cmdstring);

    msgsiz = strlen(cmdstring) + 1;
    if (msgsiz > LBUFF_SIZE) msgsiz = LBUFF_SIZE;
    strncpy(tmsgp->command, cmdstring, msgsiz);

    /*
     * send message to queue
     */
    rtn_code = msgsnd(tmsqid, tmsgp, msgsiz, msgflg);
    if (rtn_code == -1) {

```

```

    perror("msgop: msgsnd failed");
    slmstate = IDLE;
    return(rtn_code);
}

/* wait for return message */
msgtyp = 0;
if(msgrcv(rmsqid, rmsgp, MSG_SIZE, msgtyp, 0)>=0) {
    printf("\nresponse from peer received\n");
    rtn = rmsgp->code;
    if (rtn == 0) {
        fprintf(stderr, "Matlab call successful\n");
        strncpy(stateP->calib_set, rmsgp->result, LBUFF_SIZE-2);
    } else {
        slmstate = IDLE;
        fprintf(stderr, "Matlab call error\n");
        return(rtn);
    }
}

/* preprocess the raw file */
sprintf(cmdstring,"code = preprocessgc(%s/%ss/%s.d/%s.cdf, -1)",
       stateP->batch_path,stateP->samp_type,stateP->samp_id,stateP->samp_id);
fprintf(stderr,"%s\n", cmdstring);

msgs = strlen(cmdstring) + 1;
if (msgs > LBUFF_SIZE) msgs = LBUFF_SIZE;
strncpy(tmsgp->command, cmdstring, msgs);

/* */
/* send message to queue */
/* */
rtn_code = msgsnd(tmsqid, tmsgp, msgs, msgflg);
if (rtn_code == -1) {
    perror("msgop: msgsnd failed");
    slmstate = IDLE;
    return(rtn_code);
}

/* send progress message */
TKProgress( 90, "Preprocessing chromatogram.", -1);

/* wait for return message */
msgtyp = 0;
if(msgrcv(rmsqid, rmsgp, MSG_SIZE, msgtyp, 0)>=0) {
    printf("\nresponse from peer received\n");
    rtn = rmsgp->code;
    if (rtn == 0) {

```

```

        fprintf(stderr, "Matlab call successful\n");
    } else {
        slmstate = IDLE;
        fprintf(stderr, "Matlab call error\n");
        return(rtn);
    }
}

/* qa check for off scale signal */
sprintf(cmdstring,[code, min_pts, max_pts]
qa_off_scale(%s/%s,%s/software_code/matlab,%s/standards,%s/%s/%s.d/%s.cdf, 1)",
    stateP->batch_path,stateP->samp_type,
    stateP->batch_path,stateP->batch_path,
    stateP->batch_path,stateP->samp_type,stateP->samp_id,stateP->samp_id);
fprintf(stderr,"%s\n", cmdstring);
msgsz = strlen(cmdstring) + 1;
if (msgsz > LBUFF_SIZE) msgsz = LBUFF_SIZE;
strncpy(tmsgp->command, cmdstring, msgsz);
/* */
/* send message to queue */
/* */
rtn_code = msgsnd(tmsqid, tmsgp, msgsz, msgflg);
if (rtn_code == -1) {
    perror("msgop: msgsnd failed");
    slmstate = IDLE;
    return(rtn_code);
}

/* send progress message */
TKProgress( 60, "QA check for off scale signal.", -1);

/* wait for return message */
msgtyp = 0;
if(msgrcv(rmsqid, rmsgp, MSG_SIZE, msgtyp, 0)>=0) {
    printf("\nresponse from peer received\n");
    rtn = rmsgp->code;
    if (rtn == 0) {
        fprintf(stderr, "Matlab call successful\n");
        stateP->qa_scale_flag = 1;
    } else {
        slmstate = IDLE;
        fprintf(stderr, "Matlab call error\n");
        stateP->qa_scale_flag = -1;
        return(rtn);
    }
}

/* qa check for retention time markers */

```

```

        sprintf(cmdstring,"[code,           distance]
qa_retention_mark('%s/%ss','%s/software_code/matlab','%s/standards','%s/%ss/%s.d/%s.cdf',
1)",
        stateP->batch_path,stateP->samp_type,
        stateP->batch_path,stateP->batch_path,
        stateP->batch_path,stateP->samp_type,stateP->samp_id,stateP->samp_id);
fprintf(stderr,"%s\n", cmdstring);
msgsz = strlen(cmdstring) + 1;
if (msgsz > LBUFF_SIZE) msgsz = LBUFF_SIZE;
strncpy(tmsgp->command, cmdstring, msgsz);
/* */
/* send message to queue          */
/* */
rtn_code = msgsnd(tmsqid, tmsgp, msgsz, msgflg);
if (rtn_code == -1) {
    perror("msgop: msgsnd failed");
    slmstate = IDLE;
    return(rtn_code);
}

/* send progress message */
TKProgress( 30, "QA check for retention time markers.", -1);

/* wait for return message */
msgtyp = 0;
if(msgrcv(rmsqid, rmsgp, MSG_SIZE, msgtyp, 0)>=0) {
    printf("\nresponse from peer received\n");
    rtn = rmsgp->code;
    if (rtn == 0) {
        fprintf(stderr, "Retention time qa check passed\n");
        stateP->qa_rt_flag = 1;
    } else {
        slmstate = IDLE;
        fprintf(stderr, "Retention time qa check failed\n");
        stateP->qa_rt_flag = -1;
        return(rtn);
    }
}
/* determine surrogate recovery */
sprintf(cmdstring,"[code,   result,   obs,   recovery] = dim_test_surrogate('%s',   %g,
'%s/%ss/%s.d/%s.cdf', -1)",
        stateP->surr_set, stateP->surr_amt,
        stateP->batch_path,stateP->samp_type,
        stateP->samp_id,stateP->samp_id);
/*
sprintf(cmdstring,"[result, code] = runSUR('tcmx','%s/%ss/%s.d/%s.cdf', -1",
        stateP->batch_path,stateP->samp_type,stateP->samp_id,stateP->samp_id);
*/
fprintf(stderr,"%s\n", cmdstring);

```

```

msgsz = strlen(cmdstring) + 1;
if (msgsz > LBUFF_SIZE) msgsz = LBUFF_SIZE;
strncpy(tmsgp->command, cmdstring, msgsz);
/* */
/* send message to queue */ 
/* */
rtn_code = msgsnd(tmsqid, tmsgp, msgsz, msgflg);
if (rtn_code == -1) {
    perror("msgop: msgsnd failed");
    slmstate = IDLE;
    return(rtn_code);
}

/* check to see if sample type is control */
if(strcasecmp(stateP->samp_type,"control") == 0) {
    TKProgress( 15, "QA check for control sample type.", -1);
/* wait for return message */
    msgtyp = 0;
    if(msgrcv(rmsqid, rmsgp, MSG_SIZE, msgtyp, 0)>=0) {
        printf("\nresponse from peer received\n");
        sscanf(rmsgp->result,"%g",&recovery);
        stateP->surr_recover = recovery;
        rtn = rmsgp->code;
        if (rtn == 0) {
            fprintf(stderr, "Matlab call successful\n");
            stateP->qa_sur_flag = 1;
        } else {
            slmstate = IDLE;
            fprintf(stderr, "Matlab call error\n");
            stateP->qa_sur_flag = -1;
            return(rtn);
        }
    }
/* analyze the sample */
    sprintf(cmdstring,"[result, code] = dim_analyze('%s','%s', %d)",
           stateP->samp_type,stateP->samp_id, stateP->comb_mode);
    fprintf(stderr,"%s\n", cmdstring);
    strncpy(tmsgp->command, cmdstring, LBUFF_SIZE-2);
    msgsz = strlen(cmdstring) + 1;
/* */
/* send message to queue */ 
/* */
    rtn = msgsnd(tmsqid, tmsgp, msgsz, msgflg);
    if (rtn == -1) {
        perror("msgop: msgsnd failed");
        slmstate = IDLE;
        return(rtn_code);
    }
}

```

```

/* wait for return message */
msgtyp = 0;
if(msgrcv(rmsqid, rmsgp, MSG_SIZE, msgtyp, 0)>=0) {
    printf("\nresponse from peer received\n");
    rtn = rmsgp->code;
    if (rtn == 0) {
        fprintf(stderr, "Matlab call successful\n");
    } else {
        slmstate = IDLE;
        fprintf(stderr, "Matlab call error\n");
        return(rtn);
    }
}

TKProgress( 30, "QA check for calibration standard.", -1);
/* perform calibration QA check */
sprintf(cmdstring, "[code, known_conc, percent_err] = qa_cal_check(result,
'%s/%s', '%s/software_code/matlab', '%s/standards', '%s/%s/%s.d/%s.cdf', 1)",
stateP->batch_path, stateP->samp_type,
stateP->batch_path, stateP->batch_path,
stateP->batch_path, stateP->samp_type, stateP->samp_id, stateP->samp_id);
fprintf(stderr, "%s\n", cmdstring);
strncpy(tmsgp->command, cmdstring, LBUFF_SIZE-2);
msgsyz = strlen(cmdstring) + 1;

/* */
/* send message to queue */
/* */
rtn = msgsnd(tmsqid, tmsgp, msgsyz, msgflg);
if (rtn == -1) {
    perror("msgop: msgsnd failed");
    slmstate = IDLE;
    return(rtn_code);
}
} else {
/* send progress message */
TKProgress( 15, "QA check for surrogate recovery.", -1);

stateP->qa_cal_flag = 0;
}

return(rtn);
}

/* analyze gc */

int analyze(Dim_stateP stateP)
{

```

```

char cmdstring[LBUFF_SIZE+1];
extern Engine *ep;
extern char *mat_output;
Matrix *matrix_out;
double *code;
int rtn;
long msgtyp;
Cmd_msg_buf tmmsgbuf; /* message send structure */
Cmd_msg_bufP tmmsgp;
int msgsz, msgflg;
extern int tmsqid, rmsqid;
extern int slmstate;

tmmsgp = &tmmsgbuf;
tmmsgp->mtype = 1;
msgflg = 0;

if(strcasecmp(stateP->samp_type, "unknown") == 0) {
/* analyze the sample */
    sprintf(cmdstring, "[result, code] = dim_analyze(%s,%s, %d)",
           stateP->samp_type, stateP->samp_id, stateP->comb_mode);
    fprintf(stderr, "%s\n", cmdstring);
    strncpy(tmmsgp->command, cmdstring, LBUFF_SIZE-2);
    msgsz = strlen(cmdstring) + 1;

/* send progress message */
    TKProgress( 90, "Analyzing chromatogram.", -1);

/* */
/* send message to queue */
/* */
    rtn = msgsnd(tmsqid, tmmsgp, msgsz, msgflg);
    if (rtn == -1) {
        perror("msgop: msgsnd failed");
    }

} else {
/* not an "unknown" sample type */
    slmstate = IDLE;
    fprintf(stderr, "Can't analze %s sampe type\n", stateP->samp_type);
    rtn = -850;
}

return(rtn);
}

```

## **err\_code.C**

```
// ERRCODES.C
/* Written by: B. B Spencer */ 
/* Version: 08-20-96 */
// Provides description of error codes for GC DIM.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*----- This function deciphers the error codes. -----*/
extern "C" char *decipher_err_codes(int numerr,int iopt)
/* Written by: B. B Spencer */
/* Version: 08-20-96 */
// numerr is the error code to be deciphered
// iopt takes on only 3 values:
//    1 means get the function name where the error occurred
//    2 means describe the error
//    9 means list all the error codes and their definitions
{
typedef struct errcodes {
    char *funcname;
    int errnum;
    char *verbose;
} ErrCodes;

ErrCodes described[] = {

{"response_raw.m",      -10, "\tOpening of file response_raw.parm failed. Possible\n"
                           "\tcauses are: file was not created or the parmsubdir\n"
                           "\tvariable does not contain a subdirectory pointing to\n"
                           "\tthe file."},

 {"response_raw.m",      -11, "\tThe specified raw data file could not be opened."},

 {"select_data.m",        -20, "\tNo raw data exists when a request to select data was\n"
                           "\tissued."},

 {"select_data.m",        -21, "\tOpening of file select_data.parm failed. Possible\n"
                           "\tcauses are: file was not created or the parmsubdir\n"
                           "\tvariable does not contain a subdirectory pointing to\n"
                           "\tthe file."},

 {"select_data.m",        -22, "\tThe contents of select_data.parm are inconsistent, the\n"
                           "\tcauses are: file was not created or the parmsubdir\n"
                           "\tvariable does not contain a subdirectory pointing to\n"
                           "\tthe file."},
```

```

    "\tnumber of regions specified did not agree with the\n"
    "\tnumber read from the file."},

{"select_data.m", -23, "\tRegions specified in select_data.parm are overlapping."},

{"select_data.m", -24, "\tSearch for the starting subscript of a region exceeded\n"
"\tthe data array bounds."},

{"select_data.m", -25, "\tSearch for the ending subscript of a region exceeded\n"
"\tthe data array bounds."},

{"Partialpathname.m", -30, "\tThe specified subdirectory does not exist."},

{"Readfilename.m", -40, "\tThe subdirectory passed to the script does not exist."},

{"Readfilename.m", -41, "\tThe entered file name does not exist."},

{"plot_mydata.m", -50, "\tNo input data available to plot."},

{"GCAutoBaseline.m", -60, "\tNo input data available."},

{"GCAutoBaseline.m", -61, "\tOpening of file GCAutoBaseline.parm failed. Possible\n"
"\tcauses are: file was not created or the parmsubdir\n"
"\tvariable does not contain a subdirectory pointing to\n"
"\tthe file."},

{"Makeselectdataparm.m", -70, "\tAn inconsistency in the number of data regions\n"
"\tdefined in select_data.parm exists."},

{"Makeselectdataparm.m", -71, "\tThe selected data regions have overlapping retention\n"
"\t\times."},

{"savedata.m", -80, "\tNo processed data available."},

{"savedata.m", -81, "\tThe raw data file name passed to this script is not a\n"
"\tvalid UNIX file name."},

{"savedata.m", -82, "\tThe standards subdirectory passed to this script is not\n"
"\ta valid UNIX file name."},

{"savedata.m", -83, "\tThe path name for the raw data does not contain a\n"
"\tsubdirectory."},

{"savedata.m", -84, "\tThe user specified file name is not a valid UNIX file\n"
"\tname."},

{"bldcaldataset.m", -90, "\tThe number of constituents specified by the user does\n"
"\tnot agree with the value contained in the existing\n"
"\tcalfilename file name entered by the user. Error\n"

```

"\tis probably the result of an attempt to add data to\n"
 "\tcfilename from a raw data file not appropriate\n"
 "\tfor the existing calibration data information or by a\n"
 "\tchange in the regions specified in select\_data.parm file\n"
 "\tthat changed the total number of data points selected\n"
 "\tby select\_data.m."},

{"bldcaldataset.m", -91, "\tThe number of points determined for the current data\n"
 "\tarrray does not agree with the value specified in the\n"
 "\texisting calfilename file entered by the user."},

{"timeshift.m", -100, "\tNo input data available when a request to align the\n"
 "\tinput data was made."},

{"timeshift.m", -101, "\tOpening of file retentionmarkers.dat failed. Possible\n"
 "\tcauses are: file was not created or the datasubdir\n"
 "\tvariable does not contain a subdirectory pointing to\n"
 "\tthe file."},

{"timeshift.m", -102, "\tData inconsistency in file retentionmarkers.dat was\n"
 "\tdetected. The number of markers specified in the file\n"
 "\tdoes not agree with the actual number contained in\n"
 "\tthe file."},

{"timeshift.m", -103, "\tThe search for the first retention time marker failed\n"
 "\tafter the data was aligned to the mean value of the\n"
 "\tfirst retention time marker in the calibration data set."},

{"response\_set.m", -110, "\tThe file defined by calfilename could not be\n"
 "\tlocated."},

{"response\_set.m", -111, "\tThe parameter file response\_set.parm could not be\n"
 "\tlocated. Either the file was not created or the\n"
 "\tvariable parmsubdir does not point to the correct\n"
 "\tsubdirectory."},

{"response\_set.m", -112, "\tAn inconsistency between the defined data processing\n"
 "\toptions and those specified in file calfilename\n"
 "\twas detected. This will cause a fatal error if the\n"
 "\tnumber of data processing options are different."},

{"response\_set.m", -113, "\tA \"^\" character was not found as the first character\n"
 "\tin the file name for the data file being read. This\n"
 "\terror probably results from problems reported by\n"
 "\tError Code -112."},

{"response\_set.m", -114, "\tThe specified data file could not be located."},

{"response\_set.m", -115, "\tAn inconsistency between the number of points in the\n"
 "\tdata file and the number of points specified in the\n"
 "\tresponse\_set.parm file."},

```

    "\tdata file specified in file calfilename and actually\n"
    "\tread has been detected."},

{"response_set.m",      -116, "\tThe UNIX command \"basename\" from within\n"
                           "\tresponse_set.m failed while attempting to obtain the\n"
                           "\tfile name for a standard."},

{"pcamodelbld.m",       -120, "\tInvalid preprocessing option. Notify authors when\n"
                           "\tthis error occurs."},

 {"pcamodelbld.m",       -121, "\tGreater than 10% of the response matrix has zero\n"
                           "\tvariance. Suggest using an alternated data processing\n"
                           "\toption."},

 {"pcamodelbld.m",       -122, "\tA constituent in the data set is invariant. Can't model\n"
                           "\tthis data."},

 {"pcamodelbld.m",       -123, "\tThe regression equations are numerically unstable."},

 {"pcamodelbld.m",       -124, "\tThe task of building a new PCR calibration model\n"
                           "\twas not executed; the user elected to retain an\n"
                           "\texisting calibration model file based upon the entered\n"
                           "\tcalibration information data file."},

 {"integrate.m",          -130, "\tNo input data available."},

 {"integrate.m",          -131, "\tCan not locate the parameter file integration.parm in\n"
                           "\tthe specified parameter subdirectory."},

 {"integrate.m",          -132, "\tSearch for the start of the integration region exceeded\n"
                           "\tthe data array bounds."},

 {"integrate.m",          -133, "\tWorking index exceeded the data array bounds.\n"
                           "\tSuggest the number of integration intervals be\n"
                           "\tdecreased."},

 {"decimation.m",         -140, "\tNo input data available."},

 {"decimation.m",         -141, "\tCan not locate the parameter file decimation.parm in\n"
                           "\tthe specified parameter subdirectory."},

 {"decimation.m",         -142, "\tDecimation order is outside the allowed range."},

 {"psdscript.m",           -150, "\tNo input data available to process."},

 {"analbypcr.m",          -160, "\tCalibration information file could not be opened."},

 {"analbypcr.m",          -161, "\tCalibration model file could not be opened."},

```

{"analbypcr.m", -162, "\tCalibration model file does not contain a PCR model."},  
{"analbypcr.m", -163, "\tNumber of constituents in the calibration information\n"\n"\tfile and the calibration model file do not agree."},  
{"analbypcr.m", -164, "\tDisagreement between the number of points for the\n"\n"\tprocessed data listed in the calibration information\n"\n"\tfile and the calibration model file."},  
{"analbypcr.m", -165, "\tSample data array size is not consistent with the array\n"\n"\tsizes used in the calibration model."},  
{"analbypcr.m", -166, "\tCalibration information file name CANNOT end with\n"\n"\tthe suffix \".fact\"."},  
{"analbypcr.m", -167, "\tThe concentration normalized peak-area file could not\n"\n"\tbe located."},  
{"analbypcr.m", -168, "\tThe PCR model filename could not be extracted from the\n"\n"\tcomplete pathname of the PCR model file."},  
{"BaseSetData.m", -170, "\tNo input data available."},  
{"BaseSetData.m", -171, "\tParameter file, BaseSetData.parm, can not be located."},  
{"BaseSetData.m", -172, "\tSearch for a data region exceeded array bounds."},  
{"polybasefit.m", -180, "\tNo input data available to process."},  
{"polybasefit.m", -181, "\tParameter file polybasefit.m could not be located in\n"\n"\tthe currently defined parameter subdirectory."},  
{"polybasefit.m", -182, "\tThe parameter file polybasefit.m could not be opened\n"\n"\tfor saving of the new parameter values."},  
{"gen\_file\_parm.m", -190, "\tModel pathname is not a fully qualified pathname,\n"\n"\t(i.e., the pathname does not start with the \"^\"\n"\tcharacter."},  
{"gen\_file\_parm.m", -191, "\tData pathname is not a fully qualified pathname, i.e.,\n"\n"\tthe pathname does not start with the \"^\" character."},  
{"gen\_file\_parm.m", -192, "\tModel pathname is in the system root directory. This\n"\n"\tlocation is not allowed."},  
{"gen\_file\_parm.m", -193, "\tData pathname is in the system root directory. This\n"\n"\tlocation is not allowed."},  
{"gen\_file\_parm.m", -194, "\tModel pathname contains only a file name."},

```

{"gen_file_parm.m", -195, "\tData pathname contains only a file name."},  

{"gen_file_parm.m", -196, "\tThe pathname does not agree with the defined\n"
"\tsubdirectory structure."},  

{"locretentmark.m", -200, "\tNo input data available when the request to locate\n"
"\tretention time markers was made."},  

{"locretentmark.m", -201, "\tSearch for the start of the region where the retention\n"
"\ttime marker should be located exceeded the array\n"
"\tbounds."},  

 {"locretentmark.m", -202, "\tSearch for the end of the region where the retention\n"
"\ttime marker should be located exceeded the array\n"
"\tbounds."},  

 {"locretentmark.m", -203, "\tAn inconsistency between the number of markers\n"
"\tspecified in file retentionmarkers.dat and the currently\n"
"\tspecified number of markers was detected."},  

 {"locretentmark.m", -204, "\tNo raw data available."},  

 {"student.m", -210, "\tZero degrees of freedom was passed to the function;\n"
"\tthis yields an undefined t value."},  

 {"student.m", -211, "\tProbability level greater than 0.999 is not allowed;\n"
"\terrors in the calculation of t become significant at\n"
"\tsmall degrees of freedom."},  

 {"student.m", -212, "\tProbability level of zero was passed to the function."},  

 {"openGC.m", -220, "\tSpecified data file could not be opened."},  

 {"saveAIA.m", -230, "\tFilename or data arrays passed to the function are\n"
"\tnull arrays."},  

 {"saveAIA.m", -231, "\tThe root filename passed to the function is a not\n"
"\tdefined."},  

 {"fourierbaseline.m", -240, "\tNo data was passed to the function."},  

 {"fourierbaseline.m", -241, "\tParameter file \"fourierbaseline.parm\" could not be\n"
"\topened."},  

 {"fourierbaseline.m", -242, "\tThe file \"fourierbaseline.parm\" could not be opened\n"
"\tto write the new parameters."},  

 {"preprocessgc.m", -250, "\tThe UNIX command to obtain the directory name of\n"

```

"\tthe data failed."},

{"preprocessgc.m", -251, "\tThe directory for the sample does not exist."},

{"preprocessgc.m", -252, "\tThe file containing the data preprocessing options\n"\tcould not be opened."},

{"preprocessgc.m", -253, "\tThe file to save the preprocessing options could not\n"\tbe opened."},

{"analbysle.m", -260, "\tThe UNIX command to obtain the directory name of\n"\tthe data failed."},

{"analbysle.m", -261, "\tThe UNIX \"ls\" command failed or a calibration data\n"\tset file does not exist."},

{"analbysle.m", -262, "\tThe specified calibration data set file could not be\n"\topened."},

{"analbysle.m", -263, "\tMore standard files specified than are contained in\n"\tthe calibration data set file."},

{"analbysle.m", -264, "\tThe condition number of the matrix for the\n"\tcalibration model is infinite; the unknown can not be\n"\tanalyzed using the specified standards."},

{"analbysle.m", -265, "\tThe subdirectory for the specified sample does not\n"\texist."},

{"fixname.m", -270, "\tThe input filename does not contain the suffix \"CDF\"\n"\tor \"cdf\"."},

{"fixname.m", -271, "\tThe input filename contains the \"CDF\" suffix, which\n"\twas converted to \"cdf\" but the file with the new suffix\n"\tdoes not exist."},

{"fixname.m", -272, "\tThe input filename contains the \"cdf\" suffix, which\n"\twas converted to \"CDF\" but the file with the new\n"\tsuffix does not exist."},

{"runslsF.m", -280, "\tNo standard files were specified."},

{"savepcrpeaks.m", -290, "\tPeak set data is not available."},

{"savepcrpeaks.m", -291, "\tFile to save the peak retention times could not be\n"\topened."},

{"set\_dim\_path.m", -300, "\tAn error occurred while the MATLAB path was being\n"\tdetermined."},

```

{"set_dim_path.m",      -301, "\tDetermination of sample batch pathname failed."},  

{"set_dim_path.m",      -302, "\tAn error occurred while adding ..software_code/n\n"
    "\tto the MATLAB path."},  

{"set_dim_path.m",      -303, "\tAn error occurred while adding ..software_code/matlab\n"
    "\tto the MATLAB path."},  

{"set_dim_path.m",      -304, "\tAn error occurred while adding ..software_code/general\n"
    "\tto the MATLAB path."},  

{"mlrmodelbld.m",       -310, "\tNo *.peakset files were identified."},  

{"mlrmodelbld.m",       -311, "\tUNIX command to obtain the file basename from the full\n"
    "\tpathname failed."},  

{"mlrmodelbld.m",       -312, "\tUser did not select a *.peakset file."},  

{"mlrmodelbld.m",       -313, "\tUser attempted to build a new calibration model but did\n"
    "\tnot allow the overwriting of an existing model file."},  

{"mlrmodelbld.m",       -314, "\tOpening the file to save the calibration model failed\n"},  

{"mlrmodelbld.m",       -315, "\tThe peak time parameter could not be opened\n"},  

{"mlrmodelbld.m",       -316, "\tThe user did not select any peaks for the MLR model\n"},  

{"mlrmodelbld.m",       -317, "\tThe condition number of the coefficient matrix for the\n"
    "\tnormal equations suggests any MLR models using the\n"
    "\tspecified calibration data set will yield unstable results."},  

{"analbymlr.m",         -320, "\tNo *.peakset file was identified.\n"},  

{"analbymlr.m",         -321, "\tUNIX \"basename\" command failed.\n"},  

{"analbymlr.m",         -322, "\tUNIX \"dirname\" command failed.\n"},  

{"analbymlr.m",         -323, "\tThe specified directory does not exist.\n"},  

{"analbymlr.m",         -324, "\tThe specified MLR calibration file could not be opened.\n"},  

{"analbymlr.m",         -325, "\tThe specified calibration file is not an MLR model.\n"},  

{"analbymlr.m",         -326, "\tThe group file could not be opened.\n"},  

{"qa_retent_mark.m",     -330, "\tIncorrect number of parameters passed to the\n"
    "\tfunction."},
```



**INTERNAL DISTRIBUTION**

- 1.-3. M. A. Hunt
4. J. M. Jansen
5. M. L. Jernigan
- 6.-10. L. N. Klatt
11. D. W. McDonald
12. D. H. Thompson
13. K. W. Tobin
14. J. R. Younkin
- 15.-16. Central Research Library
- 17.-18. Laboratory Records
19. Laboratory Records-Record Copy
20. ORNL Patent Section
21. Y-12 Technical Library-Document Reference Center

**EXTERNAL DISTRIBUTION**

22. M. Clark, INEL Site Manager, Idaho National Engineering Laboratory, PO Box 1625 MS- 2220, Idaho Falls, ID 83415.
23. R. M. Hollen, CAA Program Coordinator, Los Alamos National Laboratory, PO Box 1663, MS-J580, Los Alamos, NM 87545.
- 24-33. K. Humphrey, Program Administrator, Los Alamos National Laboratory, PO Box 1663, MS-J580, Los Alamos, NM 87545.
34. J. Noble-Dial, ORO-DOE, 55 Jefferson, Rm219F, MS-91, Oak Ridge, TN 37831.

```

'if ac42 is absent and ac54 is present and ac60 is absent and import is medium then
sle_r_weight is zero ';
'if ac42 is absent and ac54 is present and ac60 is absent and import is low then
sle_r_weight is zero ';
'if ac42 is absent and ac54 is absent and ac60 is present and import is high then
sle_r_weight is half ';
'if ac42 is absent and ac54 is absent and ac60 is present and import is medium then
sle_r_weight is zero ';
'if ac42 is absent and ac54 is absent and ac60 is present and import is low then
sle_r_weight is zero ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is high then sle_r_weight
is one ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is medium then
sle_r_weight is half ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is low then sle_r_weight
is zero '];
sle_r_fis = parsrule(sle_r_fis, ruleTxt);
writefis(sle_r_fis,'sle_r');

```

### **startup.m**

```

P = path;
path(P, '/caa/dim/general');
P = path;
path(P, '/caa/dim/gc_dev');
P = path;
path(P, '/caa/dim/CANNS');

```

```

%Importance
sle_r_fis = addvar(sle_r_fis,'input','import',[-1 1]);

sle_r_fis = addmf(sle_r_fis, 'input',4,'low','gauss2mf',[0.34 -1.1 0.1 0.0]);
sle_r_fis = addmf(sle_r_fis, 'input',4,'medium','gauss2mf',[0.2 0.4 0.2 0.5]);
sle_r_fis = addmf(sle_r_fis, 'input',4,'high','gauss2mf',[0.1 0.9 0.34 1.1]);

%Output variables

sle_r_fis = addvar(sle_r_fis,'output','sle_r_weight',[-0.4 1.4]);

sle_r_fis = addmf(sle_r_fis, 'output',1,'zero','trimf',[-0.35 0 .35]);
sle_r_fis = addmf(sle_r_fis, 'output',1,'half','trimf',[0.15 0.5 0.85]);
sle_r_fis = addmf(sle_r_fis, 'output',1,'one','trimf',[0.65 1.0 1.35]);

ruleTxt=['if ac42 is present and ac54 is present and ac60 is present and import is high then
sle_r_weight is one ';
    'if ac42 is present and ac54 is present and ac60 is present and import is medium then
sle_r_weight is half';
    'if ac42 is present and ac54 is present and ac60 is present and import is low then
sle_r_weight is zero ';
    'if ac42 is present and ac54 is present and ac60 is absent and import is high then
sle_r_weight is one ';
    'if ac42 is present and ac54 is present and ac60 is absent and import is medium then
sle_r_weight is half';
    'if ac42 is present and ac54 is present and ac60 is absent and import is low then
sle_r_weight is zero ';
    'if ac42 is absent and ac54 is present and ac60 is present and import is high then
sle_r_weight is one ';
    'if ac42 is absent and ac54 is present and ac60 is present and import is medium then
sle_r_weight is half';
    'if ac42 is absent and ac54 is present and ac60 is present and import is low then
sle_r_weight is zero ';
    'if ac42 is present and ac54 is absent and ac60 is present and import is high then
sle_r_weight is one ';
    'if ac42 is present and ac54 is absent and ac60 is present and import is medium then
sle_r_weight is half';
    'if ac42 is present and ac54 is absent and ac60 is present and import is low then
sle_r_weight is zero ';
    'if ac42 is present and ac54 is absent and ac60 is absent and import is high then
sle_r_weight is half';
    'if ac42 is present and ac54 is absent and ac60 is absent and import is medium then
sle_r_weight is zero ';
    'if ac42 is present and ac54 is absent and ac60 is absent and import is low then
sle_r_weight is zero ';
    'if ac42 is absent and ac54 is present and ac60 is absent and import is high then
sle_r_weight is half ';

```

```

comb = find(num_results(max_index,1:noconst));

result = [];
for i = 1:length(comb)
    result = [result, ', anal_names(comb(i,:), ', ...
        sprintf('%-10.3e', num_results(max_index,comb(i))), ...
        ',sprintf('%-10.3e',-1)];
end;
result = [result, ', 'Importance = ', sprintf(' %7.3f', ...
    num_results(max_index,noconst+1))];

result = fliplr(deblank(fliplr(deblank(result))));
result = [result, setstr(0)];

return;

```

### **sle\_r\_fuzzy.m**

```

% Generate a Fuzzy inference system
% For SLE method based on raw chromatogram signal

% define the threshold for absence vs. presence for Aroclor concentration
abs_pres_thresh = 0.05;

%define the thresholds for importance - low, medium, high
imp_low_med = 0.15;
imp_med_high = 0.70;

% generate a Fuzzy inference system
sle_r_fis = newfis('sle_r','mamdani');

% add input variables
%Aroclor 1242
sle_r_fis = addvar(sle_r_fis,'input','ac42',[0 2]);

sle_r_fis = addmf(sle_r_fis, 'input',1,'absent','gauss2mf,[0.02378 0 0.03 0.02]');
sle_r_fis = addmf(sle_r_fis, 'input',1,'present','gauss2mf,[0.03 0.1 0.6795 2.2]');

%Aroclor 1254
sle_r_fis = addvar(sle_r_fis,'input','ac54',[0 2]);

sle_r_fis = addmf(sle_r_fis, 'input',2,'absent','gauss2mf,[0.02378 0 0.03 0.02]');
sle_r_fis = addmf(sle_r_fis, 'input',2,'present','gauss2mf,[0.03 0.1 0.6795 2.2]');

%Aroclor 1260
sle_r_fis = addvar(sle_r_fis,'input','ac60',[0 2]);

sle_r_fis = addmf(sle_r_fis, 'input',3,'absent','gauss2mf,[0.02378 0 0.03 0.02]');
sle_r_fis = addmf(sle_r_fis, 'input',3,'present','gauss2mf,[0.03 0.1 0.6795 2.2]');

```

```

[tmp, max_index] = max(mix_results(:,noconst+1));
for i = 1:noconst
    distance = abs(mix_results(max_index,i)- ...
        conc_mat(analyte_indices(find(analyte_indices(:,i)),i),i));
    [tmp, min_index] = min(distance);
    best_index(i) = analyte_indices(min_index,i);
end

[len, tmp] = size(mix_results);

comb = find(mixtures(max_index,:));
if length(comb) == 2
    std1 = [dim_batch_dir,'standards',gcpPath(best_index(comb(1)),:)];
    std2 = [dim_batch_dir,'standards',gcpPath(best_index(comb(2)),:)];
    [result, code] = runsleF(unkfile, mode, std1, std2);
    if code < 0
        sprintf(2,'Error in call "runsleF" generated in sle_analyze\n');
        return;
    end
    [name, conc, confid, import, code]=parse_result_string(result);
    if code < 0
        sprintf(2,'Error in call "parse_result" generated in sle_analyze\n');
        return;
    end
    mix_results(len+1, comb) = conc';
    mix_results(len+1, noconst+1) = import;

elseif length(comb) == 3
    std1 = [dim_batch_dir,'standards',gcpPath(best_index(comb(1)),:)];
    std2 = [dim_batch_dir,'standards',gcpPath(best_index(comb(2)),:)];
    std3 = [dim_batch_dir,'standards',gcpPath(best_index(comb(3)),:)];
    [result, code] = runsleF(unkfile, mode, std1, std2, std3);
    if code < 0
        sprintf(2,'Error in call "runsleF" generated in sle_analyze\n');
        return;
    end
    [name, conc, confid, import, code]=parse_result_string(result);
    if code < 0
        sprintf(2,'Error in call "parse_result" generated in sle_analyze\n');
        return;
    end
    mix_results(len+1, comb) = conc';
    mix_results(len+1, noconst+1) = import;
end

%find best overall result based on the maximum importance

num_results = [num_results; mix_results];
[tmp, max_index] = max(num_results(:,noconst+1));

```

```

%assign index of standard
best_index(i) = analyte_indices(min_index,i);
end

%iterate through the possible mixture combinations

mixtures = mix_matrix(noconst);
[len, tmp] = size(mixtures);
mix_results = zeros(len, noconst+1);

for i = 1:len
    comb = find(mixtures(i,:));
    if length(comb) == 2
        std1 = [dim_batch_dir '/standards', gcpPath(best_index(comb(1)),:)];
        std2 = [dim_batch_dir '/standards', gcpPath(best_index(comb(2)),:)];
        [result, code] = runsleF(unkfile, mode, std1, std2);
        if code < 0
            sprintf(2,'Error in call "runsleF" generated in sle_analyze\n');
            return;
        end
        [name, conc, confid, import, code]=parse_result_string(result);
        if code < 0
            sprintf(2,'Error in call "parse_result" generated in sle_analyze\n');
            return;
        end
        mix_results(i, comb) = conc';
        mix_results(i, noconst+1) = import;

    elseif length(comb) == 3
        std1 = [dim_batch_dir '/standards', gcpPath(best_index(comb(1)),:)];
        std2 = [dim_batch_dir '/standards', gcpPath(best_index(comb(2)),:)];
        std3 = [dim_batch_dir '/standards', gcpPath(best_index(comb(3)),:)];
        [result, code] = runsleF(unkfile, mode, std1, std2, std3);
        if code < 0
            sprintf(2,'Error in call "runsleF" generated in sle_analyze\n');
            return;
        end
        [name, conc, confid, import, code]=parse_result_string(result);
        if code < 0
            sprintf(2,'Error in call "parse_result" generated in sle_analyze\n');
            return;
        end
        mix_results(i, comb) = conc';
        mix_results(i, noconst+1) = import;
    end
end

% Tweak the concentration

```

```

[result, code] = runsleF(unkfile, mode, ...
    [dim_batch_dir,'/standards/',gcpPath(mid_index,:)]);
if code < 0
    sprintf(2,'Error in call "runsleF" generated in sle_analyze\n');
    return;
end
[name, conc, confid, import, code]=parse_result_string(result);
if code < 0
    sprintf(2,'Error in call "parse_result" generated in sle_analyze\n');
    return;
end
num_results((i-1)*3+2, i) = conc(1);
num_results((i-1)*3+2, noconst+1) = import;

% max conc

[result, code] = runsleF(unkfile, mode, ...
    [dim_batch_dir,'/standards/',gcpPath(max_index,:)]);
if code < 0
    sprintf(2,'Error in call "runsleF" generated in sle_analyze\n');
    return;
end
[name, conc, confid, import, code]=parse_result_string(result);
if code < 0
    sprintf(2,'Error in call "parse_result" generated in sle_analyze\n');
    return;
end
num_results((i-1)*3+3, i) = conc(1);
num_results((i-1)*3+3, noconst+1) = import;

end

% determine the best mixture combinations using best concentrations
best_index = zeros(noconst,1);

for i = 1:noconst
    %find non-zero concentrations for constituent i
    possible = find(num_results(:,i));

    %determine the maximum importance within this set
    [tmp, max_index] = max(num_results(possible,noconst+1));

    %get index for best (in term of importance) within possible
    best_index(i) = possible(max_index);

    %compute the closest standard to the reported concentration
    distance = abs(num_results(best_index(i),i)-conc_mat(:,i));
    [tmp,min_index]=min(distance(analyte_indices(find(analyte_indices(:,i)),i)));

```

```

conc_mat(1,:) = fscanf(batfile,'%f',[1,noconst]);
gcunits = fscanf(batfile,'%s',1);
if nostand > 1
    for i=2:nostand;
        tmpfile = fscanf(batfile,'%s',1);
        gcpPath = str2mat(gcpPath,tmpfile);
        conc_mat(i,:) = fscanf(batfile,'%f',[1,noconst]);
        gcunits = fscanf(batfile,'%s',1);
    end
end

fclose(batfile);
% determine the best single analyte concentration levels

% determine the concentration range on the standards

analyte_indices = zeros(nostand, noconst);
for i = 1:noconst
    tmp_vec = find(conc_mat(:,i));
    analyte_indices(1:length(tmp_vec),i) = tmp_vec;
end

num_results = zeros(3*noconst, noconst+1);

for i = 1:noconst
    [tmp, min_index] = min(conc_mat(find(conc_mat(:,i)),i));
    min_index = analyte_indices(min_index,i);
    [tmp, max_index] = max(conc_mat(find(conc_mat(:,i)),i));
    max_index = analyte_indices(max_index,i);
    mid_index = round((min_index + max_index)/2.0);

    % min conc

    [result, code] = runsleF(unkfile, mode, ...
        [dim_batch_dir,'/standards',gcpPath(min_index,:)]);
    if code < 0
        fprintf(2,'Error in call "runsleF" generated in sle_analyze\n');
        return;
    end
    [name, conc, confid, import, code]=parse_result_string(result);
    if code < 0
        fprintf(2,'Error in call "parse_result" generated in sle_analyze\n');
        return;
    end
    num_results((i-1)*3+1, i) = conc(1);
    num_results((i-1)*3+1, noconst+1) = import;

    % middle conc

```

```

fprintf(2,'Incorrect number of arguments, need unknown file and plot mode.\n');
return;
end;

% Read in *.calset file to determine the standards available.
%First find the file in the standards directory
calset_file = ls([dim_batch_dir, '/standards/*.calset']);
calset_file = fliplr(deblank(fliplr(deblank(calset_file))));
if length(calset_file) == 0
    if abs(calset_file(length(calset_file))) == 10 % new line
        calset_file = calset_file(1:length(calset_file)-1);
    end
else % error - could not find any *.calset files
    code = -270;
    fprintf(2,'Did not find any "*.calset" files in %s.\n',...
        [dim_batch_dir, '/standards/']);
    return;
end;

batfile = fopen(calset_file, 'r');
if batfile == -1 % check for valid file
    code = -900;
    fprintf(2, 'File not found.%s\n',calset_file);
    return;
end;

noconst = fscanf(batfile, '%g\n', 1);
nostand_str = fgetl(batfile);
nostand = sscanf(nostand_str, '%g', 1);

if noconst < min_const | noconst > max_const | nostand < (noconst * min_stand)
    code = -272;
    fprintf(2,'Check the contents of %s, improper number of analytes or standards.\n',...
        calset_file);
    return;
end;

for k = 1:noconst;
    dummyname = fscanf(batfile, '%s', 1);
    anal_names = str2mat(anal_names, dummyname);
end;
anal_names = anal_names(2:noconst+1,:);

conc_mat = zeros(nostand, noconst);

% read in the standard files and the corresponding concentrations
% read in first file
tmpfile = fscanf(batfile, '%s', 1);
gcpPath = str2mat(tmpfile);

```

```

%      Instrumentation and Controls Division
%      Oak Ridge National Laboratory
%      P.O. Box 2008, MS 6011
%      Oak Ridge, TN 37831-6011
% Initially created: 7/9/96
% Last Modified: 7/12/96 MAH
%
% Function call:
%
% [result, code]= sle_analyze(unkfile, mode);
%
% This function returns the following variables:
% result   A string variable containing the results. The format of
%           information within the string is:
%           analyte_name analyte_conc conc_confid
%           For multiple analytes this format is repeated for each analyte.
%           For example, assume two analytes are included in the PCR
%           calibration model, the returned string contains:
%           "1242 6.75e+02 6.23e+1 1254 -2.33e-02 1.23e-01"
% code     The integer execution code; zero indicates normal execution.
%
% The function requires the following input parameters:
% unkfile  The partial or full pathname of the unknown data file.
%           The full pathname must start from the root file system.
%           If a partial pathname is passed this must be the *.d segment;
%           the actual filename *.CDF.blr is generated from *.d.
% mode     A graphics mode flag:
%           -1 indicates plot and return immediately,
%           0 indicates do not plot, and
%           1 indicates plot and pause for 10 seconds.
%
```

```

min_const = 1; % Minimum number of constituents (analytes)
max_const = 4; % Maximum number of constituents (analytes)
min_stand = 2; % Minimum number of standards per analyte

```

```

code = 0;
result = [];
perm = [0 1 4 11];

```

```

% get batch directory
[dim_batch_dir, code] = dim_getenv('DIMBATCH_DIR');
if code ~= 0 % error condition
    return;
end

```

```

if nargin == 2
    code = -266;

```

```

'if ac42 is present and ac54 is absent and ac60 is present and import is high then
pcr_r_weight is one ';
'if ac42 is present and ac54 is absent and ac60 is present and import is medium then
pcr_r_weight is half';
'if ac42 is present and ac54 is absent and ac60 is present and import is low then
pcr_r_weight is zero ';
'if ac42 is present and ac54 is absent and ac60 is absent and import is high then
pcr_r_weight is half ';
'if ac42 is present and ac54 is absent and ac60 is absent and import is medium then
pcr_r_weight is zero ';
'if ac42 is present and ac54 is absent and ac60 is absent and import is low then
pcr_r_weight is zero ';
'if ac42 is absent and ac54 is present and ac60 is absent and import is high then
pcr_r_weight is half ';
'if ac42 is absent and ac54 is present and ac60 is absent and import is medium then
pcr_r_weight is zero ';
'if ac42 is absent and ac54 is present and ac60 is absent and import is low then
pcr_r_weight is zero ';
'if ac42 is absent and ac54 is absent and ac60 is present and import is high then
pcr_r_weight is half ';
'if ac42 is absent and ac54 is absent and ac60 is present and import is medium then
pcr_r_weight is zero ';
'if ac42 is absent and ac54 is absent and ac60 is present and import is low then
pcr_r_weight is zero ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is high then
pcr_r_weight is one ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is medium then
pcr_r_weight is half ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is low then pcr_r_weight
is zero ];
pcr_r_fis = parsrule(pcr_r_fis, ruleTxt);
writefis(pcr_r_fis,'pcr_r');

```

### sle\_analyze.m

```

function [result, code]= sle_analyze(unkfile, mode);

% FILENAME sle_analyze.m
%
% MATLAB function which determines the optimum combination of individual
% standard file to pass to the "runslsF" function based on the importance
% parameter. This function will use the *.calset file contained in the
% standards subdirectory of the current batch directory (as defined by the
% DIMBATCH_DIR).
%
% Written by: Martin Hunt

```

```

pcr_r_fis = addmf(pcr_r_fis, 'input', 1, 'present', 'gauss2mf', [0.03 0.1 0.6795 2.2]);

%Aroclor 1254
pcr_r_fis = addvar(pcr_r_fis, 'input', 'ac54', [0 2]);

pcr_r_fis = addmf(pcr_r_fis, 'input', 2, 'absent', 'gauss2mf', [0.02378 0 0.03 0.02]);
pcr_r_fis = addmf(pcr_r_fis, 'input', 2, 'present', 'gauss2mf', [0.03 0.1 0.6795 2.2]);

%Aroclor 1260
pcr_r_fis = addvar(pcr_r_fis, 'input', 'ac60', [0 2]);

pcr_r_fis = addmf(pcr_r_fis, 'input', 3, 'absent', 'gauss2mf', [0.02378 0 0.03 0.02]);
pcr_r_fis = addmf(pcr_r_fis, 'input', 3, 'present', 'gauss2mf', [0.03 0.1 0.6795 2.2]);

%Importance
pcr_r_fis = addvar(pcr_r_fis, 'input', 'import', [-1 1]);

pcr_r_fis = addmf(pcr_r_fis, 'input', 4, 'low', 'gauss2mf', [0.34 -1.1 0.1 0.0]);
pcr_r_fis = addmf(pcr_r_fis, 'input', 4, 'medium', 'gauss2mf', [0.2 0.4 0.2 0.5]);
pcr_r_fis = addmf(pcr_r_fis, 'input', 4, 'high', 'gauss2mf', [0.1 0.9 0.34 1.1]);

%Output variables

pcr_r_fis = addvar(pcr_r_fis, 'output', 'pcr_r_weight', [-0.4 1.4]);

pcr_r_fis = addmf(pcr_r_fis, 'output', 1, 'zero', 'trimf', [-0.35 0 .35]);
pcr_r_fis = addmf(pcr_r_fis, 'output', 1, 'half', 'trimf', [0.15 0.5 0.85]);
pcr_r_fis = addmf(pcr_r_fis, 'output', 1, 'one', 'trimf', [0.65 1.0 1.35]);

ruleTxt=['if ac42 is present and ac54 is present and ac60 is present and import is high then
pcr_r_weight is one ';
         'if ac42 is present and ac54 is present and ac60 is present and import is medium then
pcr_r_weight is half';
         'if ac42 is present and ac54 is present and ac60 is present and import is low then
pcr_r_weight is zero ';
         'if ac42 is present and ac54 is present and ac60 is absent and import is high then
pcr_r_weight is one ';
         'if ac42 is present and ac54 is present and ac60 is absent and import is medium then
pcr_r_weight is half';
         'if ac42 is present and ac54 is present and ac60 is absent and import is low then
pcr_r_weight is zero ';
         'if ac42 is absent and ac54 is present and ac60 is present and import is high then
pcr_r_weight is one ';
         'if ac42 is absent and ac54 is present and ac60 is present and import is medium then
pcr_r_weight is half';
         'if ac42 is absent and ac54 is present and ac60 is present and import is low then
pcr_r_weight is zero ';

```

```

'if ac42 is present and ac54 is absent and ac60 is absent and import is high then
pcr_p_weight is half ';
'if ac42 is present and ac54 is absent and ac60 is absent and import is medium then
pcr_p_weight is zero ';
'if ac42 is present and ac54 is absent and ac60 is absent and import is low then
pcr_p_weight is zero ';
'if ac42 is absent and ac54 is present and ac60 is absent and import is high then
pcr_p_weight is half ';
'if ac42 is absent and ac54 is present and ac60 is absent and import is medium then
pcr_p_weight is zero ';
'if ac42 is absent and ac54 is present and ac60 is absent and import is low then
pcr_p_weight is zero ';
'if ac42 is absent and ac54 is absent and ac60 is present and import is high then
pcr_p_weight is half ';
'if ac42 is absent and ac54 is absent and ac60 is present and import is medium then
pcr_p_weight is zero ';
'if ac42 is absent and ac54 is absent and ac60 is present and import is low then
pcr_p_weight is zero ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is high then
pcr_p_weight is one ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is medium then
pcr_p_weight is half ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is low then pcr_p_weight
is zero ']';

pcr_p_fis = parsrule(pcr_p_fis, ruleTxt);

writefis(pcr_p_fis,'pcr_p');

```

### **pcr\_r\_fuzzy.m**

```

% Generate a Fuzzy inference system
% For PCR method based on raw chromatogram signal

```

```

% define the threshold for absence vs. presence for Aroclor concentration
abs_pres_thresh = 0.05;

```

```

%define the thresholds for importance - low, medium, high
imp_low_med = 0.15;
imp_med_high = 0.70;

```

```

% generate a Fuzzy inference system
pcr_r_fis = newfis('pcr_r','mamdani');

```

```

% add input variables

```

```

%Aroclor 1242

```

```

pcr_r_fis = addvar(pcr_r_fis,'input','ac42',[0 2]);

```

```

pcr_r_fis = addmf(pcr_r_fis, 'input',1,'absent','gauss2mf,[0.02378 0 0.03 0.02]');

```

```

pcr_p_fis = addmf(pcr_p_fis, 'input', 2, 'present', 'gauss2mf', [0.03 0.1 0.6795 2.2]);

%Aroclor 1260
pcr_p_fis = addvar(pcr_p_fis, 'input', 'ac60', [0 2]);

pcr_p_fis = addmf(pcr_p_fis, 'input', 3, 'absent', 'gauss2mf', [0.02378 0 0.03 0.02]);
pcr_p_fis = addmf(pcr_p_fis, 'input', 3, 'present', 'gauss2mf', [0.03 0.1 0.6795 2.2]);

%Importance
pcr_p_fis = addvar(pcr_p_fis, 'input', 'import', [-1 1]);

pcr_p_fis = addmf(pcr_p_fis, 'input', 4, 'low', 'gauss2mf', [0.34 -1.1 0.1 0.0]);
pcr_p_fis = addmf(pcr_p_fis, 'input', 4, 'medium', 'gauss2mf', [0.2 0.4 0.2 0.5]);
pcr_p_fis = addmf(pcr_p_fis, 'input', 4, 'high', 'gauss2mf', [0.1 0.9 0.34 1.1]);

%Output variables

pcr_p_fis = addvar(pcr_p_fis, 'output', 'pcr_p_weight', [-0.4 1.4]);

pcr_p_fis = addmf(pcr_p_fis, 'output', 1, 'zero', 'trimf', [-0.35 0 0.35]);
pcr_p_fis = addmf(pcr_p_fis, 'output', 1, 'half', 'trimf', [0.15 0.5 0.85]);
pcr_p_fis = addmf(pcr_p_fis, 'output', 1, 'one', 'trimf', [0.65 1.0 1.35]);

ruleTxt='if ac42 is present and ac54 is present and ac60 is present and import is high then
pcr_p_weight is one ';
'if ac42 is present and ac54 is present and ac60 is present and import is medium then
pcr_p_weight is half';
'if ac42 is present and ac54 is present and ac60 is present and import is low then
pcr_p_weight is zero ';
'if ac42 is present and ac54 is present and ac60 is absent and import is high then
pcr_p_weight is one ';
'if ac42 is present and ac54 is present and ac60 is absent and import is medium then
pcr_p_weight is half';
'if ac42 is present and ac54 is present and ac60 is absent and import is low then
pcr_p_weight is zero ';
'if ac42 is absent and ac54 is present and ac60 is present and import is high then
pcr_p_weight is one ';
'if ac42 is absent and ac54 is present and ac60 is present and import is medium then
pcr_p_weight is half';
'if ac42 is absent and ac54 is present and ac60 is present and import is low then
pcr_p_weight is zero ';
'if ac42 is present and ac54 is absent and ac60 is present and import is high then
pcr_p_weight is one ';
'if ac42 is present and ac54 is absent and ac60 is present and import is medium then
pcr_p_weight is half';
'if ac42 is present and ac54 is absent and ac60 is present and import is low then
pcr_p_weight is zero ';

```

```

sscanf(str_result(space_indices(1):len),'%f',2);
if isempty(errmsg) ~= 1
    fprintf(2, 'Error in sscanf:%s\n',errmsg);
    code = -952;
    return;
end
conc = [conc value(1)];
confid = [confid value(2)];
nextindex = space_indices(1)+next;
space_indices = findstr(str_result(nextindex:import_start), ' ');
space_indices = space_indices + nextindex -1;
numana = numana + 1;
end

%now get importance

import = sscanf(str_result(import_start:len),'Importance = %f');

name = name(2:numana+1,:);
conc = conc';
confid = confid';

return;

pcr_p_fuzzy.m
% Generate a Fuzzy inference system
% For PCR method based on peak area input

% define the threshold for absence vs. presence for Aroclor concentration
abs_pres_thresh = 0.05;

%define the thresholds for importance - low, medium, high
imp_low_med = 0.15;
imp_med_high = 0.70;

% generate a Fuzzy inference system
pcr_p_fis = newfis('pcr_p','mamdani');

% add input variables
%Aroclor 1242
pcr_p_fis = addvar(pcr_p_fis,'input','ac42',[0 2]);

pcr_p_fis = addmf(pcr_p_fis, 'input',1,'absent','gauss2mf',[0.02378 0 0.03 0.02]);
pcr_p_fis = addmf(pcr_p_fis, 'input',1,'present','gauss2mf',[0.03 0.1 0.6795 2.2]);

%Aroclor 1254
pcr_p_fis = addvar(pcr_p_fis,'input','ac54',[0 2]);

pcr_p_fis = addmf(pcr_p_fis, 'input',2,'absent','gauss2mf',[0.02378 0 0.03 0.02]);

```

```

% name A vector of the analyte names.
%
% conc A vector of corresponding concentrations
%
% confid A vector of corresponding confidence intervals
%
% import A scalar of the importance parameter
%
% code The integer execution code; zero indicates normal execution.
%
% The function requires the following input parameters:
% str_result A string variable containing the results. The format of
% information within the string is:
% analyte_name analyte_conc conc confid importance
% For multiple analytes this format is repeated for each analyte.
% (with the exception of the importance, which is always the last
% item in the string)
% For example, assume two analytes are included in the PCR
% calibration model, the returned string contains:
% "1242 6.75e+02 6.23e+1 1254 -2.33e-02 1.23e-01"

code = 0;

len = length(str_result);
% determine where the spaces are
space_indices = findstr(str_result,' ');

% determine where Importance is

import_start = findstr(str_result,'Importance = ');

if isempty(import_start) ~= 0 % invalid format for result string
    code = -951;
    return;
end

nextindex = 1;
numana = 0;
while nextindex < import_start,
% read in analyte names
    [value, count, errmsg, next] = ...
        sscanf(str_result(nextindex:space_indices(1)),'%s');
    if isempty(errmsg) ~= 1
        fprintf(2, 'Error in sscanf:%s\n',errmsg);
        code = -952;
        return;
    end
    name = str2mat(name,value);
    [value, count, errmsg, next] = ...

```

```

'if ac42 is present and ac54 is absent and ac60 is absent and import is medium then
mlr_r_weight is zero ';
'if ac42 is present and ac54 is absent and ac60 is absent and import is low then
mlr_r_weight is zero ';
'if ac42 is absent and ac54 is present and ac60 is absent and import is high then
mlr_r_weight is half ';
'if ac42 is absent and ac54 is present and ac60 is absent and import is medium then
mlr_r_weight is zero ';
'if ac42 is absent and ac54 is present and ac60 is absent and import is low then
mlr_r_weight is zero ';
'if ac42 is absent and ac54 is absent and ac60 is present and import is high then
mlr_r_weight is half ';
'if ac42 is absent and ac54 is absent and ac60 is present and import is medium then
mlr_r_weight is zero ';
'if ac42 is absent and ac54 is absent and ac60 is present and import is low then
mlr_r_weight is zero ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is high then
mlr_r_weight is one ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is medium then
mlr_r_weight is half ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is low then mlr_r_weight
is zero ']';

mlr_r_fis = parsrule(mlr_r_fis, ruleTxt);

writefis(mlr_r_fis,'mlr_r');

```

### **parse\_result.m**

```
function [name, conc, confid, import, code]= parse_result(str_result);
```

```

% FILENAME parse_result.m
%
% MATLAB function which converts the string containing the concentration
% results and converts them to a numeric vector format.
%
% Written by: Martin Hunt
%           Instrumentation and Controls Division
%           Oak Ridge National Laboratory
%           P.O. Box 2008, MS 6011
%           Oak Ridge, TN 37831-6011
% Initially created: 7/8/96
% Last Modified: July 8, 1996 MAH
%
% Function call:
%
% [name, conc, confid, import, code]= parse_result(str_result);
%
% This function returns the following variables:
```

```

%Aroclor 1260
mlr_r_fis = addvar(mlr_r_fis,'input','ac60',[0 2]);

mlr_r_fis = addmf(mlr_r_fis, 'input',3,'absent','gauss2mf',[0.02378 0 0.03 0.02]);
mlr_r_fis = addmf(mlr_r_fis, 'input',3,'present','gauss2mf',[0.03 0.1 0.6795 2.2]);

%Importance
mlr_r_fis = addvar(mlr_r_fis,'input','import',[-1 1]);

mlr_r_fis = addmf(mlr_r_fis, 'input',4,'low','gauss2mf',[0.34 -1.1 0.1 0.0]);
mlr_r_fis = addmf(mlr_r_fis, 'input',4,'medium','gauss2mf',[0.2 0.4 0.2 0.5]);
mlr_r_fis = addmf(mlr_r_fis, 'input',4,'high','gauss2mf',[0.1 0.9 0.34 1.1]);

%Output variables

mlr_r_fis = addvar(mlr_r_fis,'output','mlr_r_weight',[-0.4 1.4]);

mlr_r_fis = addmf(mlr_r_fis, 'output',1,'zero','trimf',[-0.35 0 .35]);
mlr_r_fis = addmf(mlr_r_fis, 'output',1,'half','trimf',[0.15 0.5 0.85]);
mlr_r_fis = addmf(mlr_r_fis, 'output',1,'one','trimf',[0.65 1.0 1.35]);

ruleTxt=['if ac42 is present and ac54 is present and ac60 is present and import is high then
mlr_r_weight is one ';
         'if ac42 is present and ac54 is present and ac60 is present and import is medium then
mlr_r_weight is half';
         'if ac42 is present and ac54 is present and ac60 is present and import is low then
mlr_r_weight is zero ';
         'if ac42 is present and ac54 is present and ac60 is absent and import is high then
mlr_r_weight is one ';
         'if ac42 is present and ac54 is present and ac60 is absent and import is medium then
mlr_r_weight is half';
         'if ac42 is present and ac54 is present and ac60 is absent and import is low then
mlr_r_weight is zero ';
         'if ac42 is absent and ac54 is present and ac60 is present and import is high then
mlr_r_weight is one ';
         'if ac42 is absent and ac54 is present and ac60 is present and import is medium then
mlr_r_weight is half';
         'if ac42 is absent and ac54 is present and ac60 is present and import is low then
mlr_r_weight is zero ';
         'if ac42 is present and ac54 is absent and ac60 is present and import is high then
mlr_r_weight is one ';
         'if ac42 is present and ac54 is absent and ac60 is present and import is medium then
mlr_r_weight is half';
         'if ac42 is present and ac54 is absent and ac60 is present and import is low then
mlr_r_weight is zero ';
         'if ac42 is present and ac54 is absent and ac60 is absent and import is high then
mlr_r_weight is half ';

```

```

mlr_p_fis = addmf(mlr_p_fis, 'input', 2, 'medium', 'gauss2mf', [0.2 0.4 0.2 0.5]);
mlr_p_fis = addmf(mlr_p_fis, 'input', 2, 'high', 'gauss2mf', [0.1 0.9 0.34 1.1]);

%Output variables

mlr_p_fis = addvar(mlr_p_fis, 'output', 'mlr_p_weight', [-0.4 1.4]);

mlr_p_fis = addmf(mlr_p_fis, 'output', 1, 'zero', 'trimf', [-0.35 0 .35]);
mlr_p_fis = addmf(mlr_p_fis, 'output', 1, 'half', 'trimf', [0.15 0.5 0.85]);
mlr_p_fis = addmf(mlr_p_fis, 'output', 1, 'one', 'trimf', [0.65 1.0 1.35]);

ruleTxt=['if anal is present and import is high then mlr_p_weight is one ';
          'if anal is present and import is medium then mlr_p_weight is half';
          'if anal is present and import is low then mlr_p_weight is zero ';
          'if anal is absent and import is high then mlr_p_weight is one ';
          'if anal is absent and import is medium then mlr_p_weight is half';
          'if anal is absent and import is low then mlr_p_weight is zero '];

mlr_p_fis = parsrule(mlr_p_fis, ruleTxt);

writefis(mlr_p_fis, 'mlr_p');

```

### **mlr\_r\_fuzzy.m**

```

% Generate a Fuzzy inference system
% For MLR method based on raw chromatogram signal

% define the threshold for absence vs. presence for Aroclor concentration
abs_pres_thresh = 0.05;

%define the thresholds for importance - low, medium, high
imp_low_med = 0.15;
imp_med_high = 0.70;

% generate a Fuzzy inference system
mlr_r_fis = newfis('mlr_r', 'mamdani');

% add input variables
%Aroclor 1242
mlr_r_fis = addvar(mlr_r_fis, 'input', 'ac42', [0 2]);

mlr_r_fis = addmf(mlr_r_fis, 'input', 1, 'absent', 'gauss2mf', [0.02378 0 0.03 0.02]);
mlr_r_fis = addmf(mlr_r_fis, 'input', 1, 'present', 'gauss2mf', [0.03 0.1 0.6795 2.2]);

%Aroclor 1254
mlr_r_fis = addvar(mlr_r_fis, 'input', 'ac54', [0 2]);

mlr_r_fis = addmf(mlr_r_fis, 'input', 2, 'absent', 'gauss2mf', [0.02378 0 0.03 0.02]);
mlr_r_fis = addmf(mlr_r_fis, 'input', 2, 'present', 'gauss2mf', [0.03 0.1 0.6795 2.2]);

```

```

%
% code      The integer execution code; zero indicates normal execution.
%
% The function requires the following input parameters:
% noconst   An integer specifying the number of constituents

code = 0;

if noconst == 1
    matrix = [];
elseif noconst == 2
    matrix = [1 1];
elseif noconst == 3
    matrix = [1 1 0; 1 0 1; 0 1 1; 1 1 1];
elseif noconst == 4
    matrix = [1 1 0 0; 1 0 1 0; 1 0 0 1; ...
               0 1 1 0; 0 1 0 1; ...
               0 0 1 1; ...
               1 1 1 0; 1 0 1 1; ...
               0 1 1 1; 1 1 1 1];
end
return;

```

### **mlr\_p\_fuzzy.m**

```

% Generate a Fuzzy inference system
% For MLR method based on peak area input

% define the threshold for absence vs. presence for Aroclor concentration
abs_pres_thresh = 0.05;

%define the thresholds for importance - low, medium, high
imp_low_med = 0.15;
imp_med_high = 0.70;

% generate a Fuzzy inference system
mlr_p_fis = newfis('mlr_p','mamdani');

% add input variables
%Analyte
mlr_p_fis = addvar(mlr_p_fis,'input','anal',[0 2]);

mlr_p_fis = addmf(mlr_p_fis, 'input',1,'absent','gauss2mf',[0.02378 0 0.03 0.02]);
mlr_p_fis = addmf(mlr_p_fis, 'input',1,'present','gauss2mf',[0.03 0.1 0.6795 2.2]);

%Importance
mlr_p_fis = addvar(mlr_p_fis,'input','import',[-1 1]);

mlr_p_fis = addmf(mlr_p_fis, 'input',2,'low','gauss2mf',[0.34 -1.1 0.1 0.0]);

```

```

'if ac42 is present and ac54 is present and ac60 is present then lr_p_weight is zero
';
'if ac42 is present and ac54 is present and ac60 is absent then lr_p_weight is zero
';
'if ac42 is absent and ac54 is present and ac60 is present then lr_p_weight is zero
';
'if ac42 is present and ac54 is absent and ac60 is present and import is high then
lr_p_weight is half ';
'if ac42 is present and ac54 is absent and ac60 is present and import is medium then
lr_p_weight is zero ';
'if ac42 is present and ac54 is absent and ac60 is present and import is low then
lr_p_weight is zero ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is high then lr_p_weight
is one ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is medium then
lr_p_weight is half ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is low then lr_p_weight
is zero ';
lr_p_fis = parsrule(lr_p_fis, ruleTxt);

writefis(lr_p_fis,'lr_p');

```

### **mix\_matrix.m**

```

function [matrix, code]= mix_matrix(noconst);

% FILENAME mix_matrix.m
%
% MATLAB function which generates a matrix with the possible mixture
% combinations given the number of constituents. The matrix contains the
% same number of columns as constituents and the number of rows corresponds
% to the total possible combinations
%
% Written by: Martin Hunt
%           Instrumentation and Controls Division
%           Oak Ridge National Laboratory
%           P.O. Box 2008, MS 6011
%           Oak Ridge, TN 37831-6011
% Initially created: 7/8/96
% Last Modified: July 8, 1996 MAH
%
% Function call:
%
% [matrix, code]= mix_matrix(noconst);
%
% This function returns the following variables:
%   matrix  A matrix of combinations, a one indicates inclusion of the
%           corresponding column (constituent)

```

```

lr_p_fis = addmf(lr_p_fis, 'input',1,'present','gauss2mf',[0.03 0.1 0.6795 2.2]);

%Aroclor 1254
lr_p_fis = addvar(lr_p_fis,'input','ac54',[0 2]);

lr_p_fis = addmf(lr_p_fis, 'input',2,'absent','gauss2mf',[0.02378 0 0.03 0.02]);
lr_p_fis = addmf(lr_p_fis, 'input',2,'present','gauss2mf',[0.03 0.1 0.6795 2.2]);

%Aroclor 1260
lr_p_fis = addvar(lr_p_fis,'input','ac60',[0 2]);

lr_p_fis = addmf(lr_p_fis, 'input',3,'absent','gauss2mf',[0.02378 0 0.03 0.02]);
lr_p_fis = addmf(lr_p_fis, 'input',3,'present','gauss2mf',[0.03 0.1 0.6795 2.2]);

%Importance
lr_p_fis = addvar(lr_p_fis,'input','import',[-1 1]);

lr_p_fis = addmf(lr_p_fis, 'input',4,'low','gauss2mf',[0.34 -1.1 0.1 0.0]);
lr_p_fis = addmf(lr_p_fis, 'input',4,'medium','gauss2mf',[0.2 0.4 0.2 0.5]);
lr_p_fis = addmf(lr_p_fis, 'input',4,'high','gauss2mf',[0.1 0.9 0.34 1.1]);

%Output variables

lr_p_fis = addvar(lr_p_fis,'output','lr_p_weight',[-0.4 1.4]);

lr_p_fis = addmf(lr_p_fis, 'output',1,'zero','trimf',[-0.35 0 .35]);
lr_p_fis = addmf(lr_p_fis, 'output',1,'half','trimf',[0.15 0.5 0.85]);
lr_p_fis = addmf(lr_p_fis, 'output',1,'one','trimf',[0.65 1.0 1.35]);

ruleTxt=['if ac42 is present and ac54 is absent and ac60 is absent and import is high then
lr_p_weight is one ';
         'if ac42 is present and ac54 is absent and ac60 is absent and import is medium then
lr_p_weight is half ';
         'if ac42 is present and ac54 is absent and ac60 is absent and import is low then lr_p_weight
is zero ';
         'if ac42 is absent and ac54 is present and ac60 is absent and import is high then lr_p_weight
is one ';
         'if ac42 is absent and ac54 is present and ac60 is absent and import is medium then
lr_p_weight is half ';
         'if ac42 is absent and ac54 is present and ac60 is absent and import is low then lr_p_weight
is zero ';
         'if ac42 is absent and ac54 is absent and ac60 is present and import is high then lr_p_weight
is one ';
         'if ac42 is absent and ac54 is absent and ac60 is present and import is medium then
lr_p_weight is half ';
         'if ac42 is absent and ac54 is absent and ac60 is present and import is low then lr_p_weight
is zero ';

```

```
[dim_batch_dir,'/software_code/matlab'],...  
[dim_batch_dir,'/standards'],...  
source, 1);
```

```
end % do not run qa calibration check
```

```
return;
```

### fis\_range.m

```
function [fis_input,code]= fis_range(input, fis)
```

```
code = 0;  
fis_input = input;  
num_inputs = getfis(fis, 'numinputs');  
  
if length(input) == num_inputs  
    for i = 1:num_inputs  
        input_range = getfis(fis,'input',i,'range');  
        if input(i) < input_range(1)  
            fis_input(i) = input_range(1);  
        elseif input(i) > input_range(2)  
            fis_input(i) = input_range(2);  
        end  
    end  
else  
    code = -1;  
end  
return
```

### lr\_p\_fuzzy.m

```
% Generate a Fuzzy inference system  
% Linear regression using peak area as input
```

```
% define the threshold for absence vs. presence for Aroclor concentration  
abs_pres_thresh = 0.05;
```

```
%define the thresholds for importance - low, medium, high
```

```
imp_low_med = 0.15;  
imp_med_high = 0.70;
```

```
% generate a Fuzzy inference system  
lr_p_fis = newfis('lr_p','mamdani');
```

```
% add input variables  
%Aroclor 1242
```

```
lr_p_fis = addvar(lr_p_fis,'input','ac42',[0 2]);
```

```
lr_p_fis = addmf(lr_p_fis, 'input',1,'absent','gauss2mf',[0.02378 0 0.03 0.02]);
```

```

dim_batch_dir = dim_getenv('DIMBATCH_DIR');

% preprocess sample

samp_dir = [dim_batch_dir,'/',sample_type,'s',sample_id,'.d'];
source=[samp_dir,'/',sample_id,'.cdf']
code = preprocessgc(source, -1);

if code < 0
    return;
end;

% perform off scale check

[code, min_pts, max_pts] = qa_off_scale([dim_batch_dir,'/',sample_type,'s'],...
    [dim_batch_dir,'/software_code/matlab'], ...
    [dim_batch_dir,'/standards'],...
    source, 1);

if code < 0
    return;
end

% perform rt marker check

[code, distance] = qa_retention_mark([dim_batch_dir,'/',sample_type,'s'],...
    [dim_batch_dir,'/software_code/matlab'], ...
    [dim_batch_dir,'/standards'],...
    source, 1);

if code < 0
    return;
end

%determine sample type

if strcmp(sample_type,'control') % must run qa calibration check

%analyze sample
[result, code] = dim_analyze(sample_type, sample_id, comb_mode);

if code < 0
    return;
end

%check calibration

[code, known_conc, percent_err] = qa_cal_check(result, ...
    [dim_batch_dir,'/',sample_type,'s'],...

```

```

end

result = sprintf('%g',recovery);

set(0,'Diary','off');
return;

dim_validate.m
function code = dim_validate(sample_type, sample_id, comb_mode);

% FILENAME dim_validate.m
%
% MATLAB function which calls the necessary qa/validation functions. Will
% return on first failure.
%
% Written by: Martin Hunt
%           Instrumentation and Controls Division
%           Oak Ridge National Laboratory
%           P.O. Box 2008, MS 6011
%           Oak Ridge, TN 37831-6011
% Initially created: 9/18/96
% Last Modified:
%
% Function call:
%
% code = dim_validate(sample_type, sample_id, comb_mode);
%
% This function returns the following variables:
% code      The integer execution code; zero indicates normal execution.
%
% The function requires the following input parameters:
% sample_type A string containing one of the valid AIA sample types
%               (standard, unknown, control, blank) corresponding to the
%               sample.
%
% sample_id   A text string specifying the id of the sample to be analyzed.
%               The file should be located in the "sample_type" subdirectory of
%               the current batch processing directory (as defined by the
%               return from "dim_getenv('DIMBATCH_DIR')").
%
% comb_mode   The type of combination of the results from the various methods
%               (might be just a single method). Numeric value based on
%               enumerated list defined in "general/dim_slm.h"

% initialize variables

code = 0;

```

```

    code = -331;
    fprintf(1, '%s%d\n', 'Error code = ', code );
    return;
else;
    ltolerance = fscanf(parmfileid, '%g\n', 1);
    utolerance = fscanf(parmfileid, '%g\n', 1);
    fclose(parmfileid);
end;

% determine the recovery

samp_dir = sample;

[results, code] = runSUR(name, samp_dir, plot);

% setup diary
%
dir_index = max(findstr(sample,'/')) - 1;
diaryfile = sample(1:dir_index);
diaryfile = [diaryfile, '/', 'surrogate_result.diary'];
set(0,'DiaryFile',diaryfile);
set(0,'Diary','on');

if code ~= 0
    fprintf(1, '%s%d\n', 'Error code = ', code);
    set(0,'Diary','off');return ;
end

% parse result string

[analyte_names, conc, conf, importance, code] = ...
    parse_result_string(results);

if code ~= 0
    fprintf(1, '%s%d\n', 'Error code = ', code);
    set(0,'Diary','off');return ;
end

fprintf(1,'Recovery limits are %f and %f percent\n',ltolerance,utolerance);
recovery = (conc(1) / amt) *100;
fprintf(1,'Specified amount of %s is %g ng\n',name,amt);
fprintf(1,'Actual recovery of %s is %g percent\n',name,recovery);
obs = conc(1);

if (recovery > ltolerance) & (recovery < utolerance)
    code = 0;
else
    code = -1;

```

```

% This function returns the following variables:
%   code      The integer error code, 0 indicates surrogate recovery
%             is within acceptable tollerances, negative value
%             indicates a failure (recovery low or high).
%
%   obs       The measured concentration of the surrogate in ng.
%
%   recovery   The percent recovery of the surrogate.
%
% This script requires the following input variables:
%   name      The name of the surrogate in the form of a string
%
%   amt       The true amount of the surrogate in the sample in ng
%
%   sample    A text string specifying the full pathname to the sample
%             to be analyzed.
%
%   plot      A graphics mode flag:
%             <0 indicates plot and return immediately,
%             =0 indicates do not plot, and
%             >0 indicates plot and pause the specified seconds.
%
code = -1;
obs = -1;
recovery = 0;
result = [];

%
% Check the number of input arguments.
%
if nargin ~= 4;
    fprintf(1, '%s\n', 'Wrong number of arguments passed to the function.');
    fprintf(1, '%s%d\n', 'Error code = ', code);
    return;
end;

%
% Read the recovery limits from the parameter file.
% The error limit is defined in terms of the percentage of the
% surrogate recovered
%
parmsubdir = [dim_getenv('DIMBATCH_DIR'), '/software_code/matlab'];
datasubdir = [dim_getenv('DIMBATCH_DIR'), '/unknowns'];
parmfile = [parmsubdir, '/', 'surrogate.parm'];
parmfileid = fopen(parmfile, 'r');
if(parmfileid == -1);
    fprintf(1, '%s%s%s\n', 'File: ', parmfile, ' could not be opened for reading.');

```

```

        sprintf(1, '%s%s%s\n', 'Transfer of files to ', dimenv, ' is complete.');
%
% Set the group membership on all the files in the new batch subdirectory.
%
[unixstatus, unixresult] = unix(['chgrp -R ', caagroup, ' ', dimenv]);
if(unixstatus ~= 0);
    sprintf(1, '%s\n', 'UNIX "chgrp" command failed.');
    code = -576;
    return;
end;
sprintf(1, '%s%s%s%s\n', 'All files in ', dimenv, ...
    ' have been assigned to group: ', caagroup);

%
% Move to the ../software_code/mathlab subdirectory and set the matlab path to
% include the new directories.
%
eval(['cd ', dimenv, software_code, '/matlab']);

end;

code = set_dim_path;
if code < 0
    sprintf(1, 'Error in set_dim_path:%d\n');
    return;
end;
end;
return;

```

### **dim\_test\_surrogate.m**

```

function [code, result, obs, recovery] = dim_test_surrogate(name, amt, sample, plot)
% FILENAME dim_test_surrogate.m
%
% MATLAB function which calls the necessary function to determine the level
% of surrogate recovery (runSUR.m). The function returns a pass/fail code base
% on the tolerances specified in the file "surrogate.prm"
%
% Written by: Martin A. Hunt
%           Instrumentation & Controls Divison
%           Oak Ridge National Laboratory
%           P.O. Box 2008
%           Oak Ridge, TN 37831-6011
%           Initially created: May 1, 1997
%           Last modified: May 5, 1997
%
% Function call:
%
% [code, obs, recovery] = dim_test_surrogate(name, amt, sample, plot)
%
```

```

[unixstatus, unixresult] = unix(['chmod 660 ', dimenv,
software_code,'/nn/*']);
if(unixstatus ~= 0);
    fprintf(1, '%s\n', 'UNIX "chmod" command failed while
changing permissions on the ./nn files.');
    code = -572;
    return;
end;
end;

%
% Now copy the general files.
%
[unixstatus, output] = unix(['ls ', generalcode]);
if(unixstatus ~= 0);
    fprintf(1, '%s\n', 'UNIX "ls" command failed while obtain a listing of the
source GENERAL code and parameter files.');
    code = -573;
    return;
end;
if(isempty(output) == 1);
    fprintf(1, '%s%s%s\n', 'Subdirectory ', generalcode, ' is empty!');
    fprintf(1, '%s\n', 'No files will be copied.');
else;
    fprintf(1, '%s%s%s%s\n',...
        'Source "general" code is being copied to the',...
        dimenv, software_code, '/general subdirectory.');
    nfiles = length(findstr(abs(10), output));
    for k = 1:nfiles;
        nullindex = findstr(abs(10), output);
        filename = output(1:nullindex(1) - 1);
        output = output(nullindex(1) + 1:length(output));
        [unixstatus, unixresult] = unix(['cp ', generalcode, ...
            '/', filename, '',...
            dimenv, software_code, '/general/', filename]);
        if(unixstatus ~= 0);
            fprintf(1, '%s\n', 'UNIX "cp" command failed.');
            code = -574;
            return;
        end;
    end;
    [unixstatus, unixresult] = unix(['chmod 660 ', dimenv,
software_code,'/general/*']);
    if(unixstatus ~= 0);
        fprintf(1, '%s\n', 'UNIX "chmod" command failed while
changing permissions on the general files.');
        code = -575;
        return;
    end;
end;

```

```

        sprintf(1, '%s\n', 'UNIX "cp" command failed.');
        code = -568;
        return;
    end;
end;
[unixstatus, unixresult] = unix(['chmod 660 ',dimenv,'/pcr/*']);
if(unixstatus ~= 0);
    sprintf(1, '%s\n', 'UNIX "chmod" command failed while changing
permission on parameter files.');
    code = -569;
    return;
end;
[unixstatus, unixresult] = unix(['chmod 660 ',dimenv,'/mlr/*']);
if(unixstatus ~= 0);
    sprintf(1, '%s\n', 'UNIX "chmod" command failed while changing
permission on parameter files.');
    code = -569;
    return;
end;
%
% Now copy the neural network code and any parameter files.
%
[unixstatus, output] = unix(['ls ', nncode]);
if(unixstatus ~= 0);
    sprintf(1, '%s\n', 'UNIX "ls" command failed while obtain a listing of the
source NEURAL NETWORK code and parameter files.');
    code = -570;
    return;
end;
if(isempty(output) == 1);
    sprintf(1, '%s%s%s\n', 'Subdirectory ', nncode, ' is empty!');
    sprintf(1, '%s\n', 'No files will be copied.');
else;
    sprintf(1, '%s%s%s%s\n', 'Source NN code is being copied to the',...
        dimenv, software_code, '/nn subdirectory');
    nofiles = length(findstr(abs(10), output));
    for k = 1:nofiles;
        nullindex = findstr(abs(10), output);
        filename = output(1:nullindex(1) - 1);
        output = output(nullindex(1) + 1:length(output));
        [unixstatus, unixresult] = unix(['cp ', nncode, ...
            '/', filename, '',...
            dimenv, software_code, '/nn/', filename]);
        if(unixstatus ~= 0);
            sprintf(1, '%s\n', 'UNIX "cp" command failed.');
            code = -571;
            return;
        end;
    end;

```

```

% yields this result independent of the permissions on the source files.
%
[unixstatus, unixresult] = unix(['chmod 550 ', dimenv, software_code,
'/matlab/*']);
    if(unixstatus ~= 0);
        fprintf(1, '%s\n', 'UNIX "chmod" failed while setting
read and execute file permissions in ../matlab subdirectory.');
        code = -567;
        return;
    end;
[unixstatus, unixresult] = unix(['chmod 440 ', dimenv, software_code,
'/matlab/*.m']);
    if(unixstatus ~= 0);
        fprintf(1, '%s\n', 'UNIX "chmod" failed while setting
read only file permissions in ../matlab subdirectory.');
        code = -567;
        return;
    end;
[unixstatus, unixresult] = unix(['chmod 660 ', dimenv, software_code,
'/matlab/*.parm']);
    if(unixstatus ~= 0);
        fprintf(1, '%s\n', 'UNIX "chmod" failed while setting
read and write file permissions in ../matlab subdirectory.');
        code = -567;
        return;
    end;

end;
%
% Put copies of two parameter files into the ../per and ../mlr subdirectories.
% The copies in ../matlab will be the default files.
%
parmfile(1,:) = 'decimation.parm';
parmfile(2,:) = 'integration.parm';
fprintf(1, '%s\n', 'Copying the relevant parameter files into ../per and ../mlr.');
for k = 1:min(size(parmfile));
    filename = deblank(parmfile(k,:));
    [unixstatus, unixresult] = unix(['cp ', matlabcode, ...
        '/', filename, ...
        ' ', dimenv, '/per/', filename]);
    if(unixstatus ~= 0);
        fprintf(1, '%s\n', 'UNIX "cp" command failed.');
        code = -568;
        return;
    end;
    [unixstatus, unixresult] = unix(['cp ', matlabcode, ...
        '/', filename, ...
        ' ', dimenv, '/mlr/', filename]);
    if(unixstatus ~= 0);

```

```

        code = -563;
        return;
    end;
    [unixstatus, unixresult] = unix(['chmod -R 2770 ',dimenv, software_code]);
    if(unixstatus ~= 0);
        fprintf(1, '%s\n', 'UNIX "chmod" command failed while creating the
new batch subdirectory.');
        code = -564;
        return;
    end;
    fprintf(1, '%s%s%s\n', 'Creation of the directory structure for ', dimenv, ' is
complete.');
%
% Copy the relevant code from the source to the destination directories.
% Copy the MATLAB code and parameter files.
%
[unixstatus, output] = unix(['ls ', matlabcode]);
if(unixstatus ~= 0);
    fprintf(1, '%s\n', 'UNIX "ls" command failed while obtaining a listing of
the source MATLAB code and parameter files.');
    code = -565;
    return;
end;
if(isempty(output) == 1);
    fprintf(1, '%s%s%s\n', 'Subdirectory ', matlabcode, ' is empty!');
    fprintf(1, '%s\n', 'No files will be copied.');
else;
    fprintf(1, '%s%s%s%s\n',...
        'Source MATLAB code is being copied to the',...
        dimenv, software_code, '/matlab subdirectory.');
    nofiles = length(findstr(abs(10), output));
    for k = 1:nofiles;
        nullindex = findstr(abs(10), output);
        filename = output(1:nullindex(1)-1);
        output = output(nullindex(1)+1:length(output));
        [unixstatus, unixresult] = unix(['cp ', matlabcode, ...
            '/', filename, '',...
            dimenv, software_code, '/matlab/', filename]);
        if(unixstatus ~= 0);
            fprintf(1, '%s\n', 'UNIX "cp" command failed.');
            code = -566;
            return;
        end;
    end;
%
% Set the file permissions to protect the code in the batch subdirectory.
% Only the user and members of the group will have permissions.
% MATLAB M-files have read permission, binary files have read and execute permissions,
% and parameter files have read write permissions. The permissions coding

```

```

[unixstatus, unixresult] = unix(['mkdir ', dimenv, '/mlr']);
if(unixstatus ~= 0);
    fprintf(1, '%s\n', 'UNIX "mkdir" command failed while creating the
"mlr" subdirectory.');
    code = -557;
    return;
end;
[unixstatus, unixresult] = unix(['mkdir ', dimenv, '/nn']);
if(unixstatus ~= 0);
    fprintf(1, '%s\n', 'UNIX "mkdir" command failed while creating the "nn"
subdirectory.');
    code = -558;
    return;
end;
[unixstatus, unixresult] = unix(['chmod 2770 ',dimenv,'/*']);
if(unixstatus ~= 0);
    fprintf(1, '%s\n', 'UNIX "chmod" command failed while creating the
new batch subdirectory.');
    code = -559;
    return;
end;
%
% Create the software subdirectories.
%
[unixstatus, unixresult] = unix(['mkdir ', dimenv, software_code]);
if(unixstatus ~= 0);
    fprintf(1, '%s\n', 'UNIX "mkdir" command failed while creating the
"software_code" subdirectory.');
    code = -560;
    return;
end;
[unixstatus, unixresult] = unix(['mkdir ', dimenv, software_code, '/matlab']);
if(unixstatus ~= 0);
    fprintf(1, '%s\n', 'UNIX "mkdir" command failed while creating the
"matlab" software subdirectory.');
    code = -561;
    return;
end;
[unixstatus, unixresult] = unix(['mkdir ', dimenv, software_code, '/nn']);
if(unixstatus ~= 0);
    fprintf(1, '%s\n', 'UNIX "mkdir" command failed while creating the
"neural network" software subdirectory.');
    code = -562;
    return;
end;
[unixstatus, unixresult] = unix(['mkdir ', dimenv, software_code, '/general']);
if(unixstatus ~= 0);
    fprintf(1, '%s\n', 'UNIX "mkdir" command failed while creating the
"general" software subdirectory.');

```

```

        mystring, ' is undefined.');
    return;
else;
    [unixstatus, unixresult] = unix(['chmod 2770 ',dimenv]);
    if(unixstatus ~= 0);
        fprintf(1, "%s\n", 'UNIX "chmod" command failed while creating
the new batch subdirectory.');
        code = -551;
        return;
    end;
end;
%
% Create the data subdirectories for the new batch.
%
[unixstatus, unixresult] = unix(['mkdir ', dimenv, '/standards']);
if(unixstatus ~= 0);
    fprintf(1, "%s\n", 'UNIX "mkdir" command failed while creating the
"standards" subdirectory.');
    code = -552;
    return;
end;
[unixstatus, unixresult] = unix(['mkdir ', dimenv, '/unknowns']);
if(unixstatus ~= 0);
    fprintf(1, "%s\n", 'UNIX "mkdir" command failed while creating the
"unknowns" subdirectory.');
    code = -553;
    return;
end;
[unixstatus, unixresult] = unix(['mkdir ', dimenv, '/controls']);
if(unixstatus ~= 0);
    fprintf(1, "%s\n", 'UNIX "mkdir" command failed while creating the
"controls" subdirectory.');
    code = -554;
    return;
end;
[unixstatus, unixresult] = unix(['mkdir ', dimenv, '/blanks']);
if(unixstatus ~= 0);
    fprintf(1, "%s\n", 'UNIX "mkdir" command failed while creating the
"blanks" subdirectory.');
    code = -555;
    return;
end;
[unixstatus, unixresult] = unix(['mkdir ', dimenv, '/pcr']);
if(unixstatus ~= 0);
    fprintf(1, "%s\n", 'UNIX "mkdir" command failed while creating the
"pcr" subdirectory.');
    code = -556;
    return;
end;

```

```

fprintf(1, '%s%s\n', 'The current SAMPLE BATCH DIRECTORY is: ', dimenv);
%
% Create batch directory environment variable
%
dim_setenv('DIMBATCH_DIR', dimenv);
[dirstatus, code] = direxist(dimenv);
if(code ~= 0);
    fprintf(1,'Error in the "direxist" function\n');
    return;
end;
if(dirstatus == 2);
    fprintf(1, '%s%s%s\n', 'A directory structure for ', dimenv, ' exists');
    fprintf(1, '%s%s\n', 'The GC DIM will run using code from ', dimenv);
    [dirstatus, code] = direxist(['dimenv software_code '/matlab']);
    if(code ~= 0);
        fprintf(1,'Error in the "direxist" function\n');
        return;
    end
    if(dirstatus ~= 2);
        sprintf(1,'subdirectory %s%s/matlab does not exist\n', ...
            dimenv, software_code);
        code = -980;
        return;
    end;

    eval(['cd ', dimenv, software_code, '/matlab']);
else;
%
% Create the new batch subdirectory.
%
fprintf(1, '%s\n', 'The directory structure for the new batch is being established.');
[unixstatus, unixresult] = unix(['mkdir ', dimenv]);
if(unixstatus ~= 0);
    fprintf(1, '%s\n', 'UNIX "mkdir" command failed while creating the new
batch subdirectory.');
    mystring = fliplr(dimenv);
    while(mystring(1) ~= '/');
        mystring = mystring(2:length(mystring));
        if(length(mystring) == 0);
            sprintf(1, '%s%s%s\n', ...
                'The partial pathname for the batch
subdirectory',...
                dimenv, ' is not defined properly.');
            code = -550;
            return;
        end;
    end;
    mystring = fliplr(mystring(2:length(mystring)));
    sprintf(1, '%s%s%s\n', 'A probable cause: Partial pathname',...

```

```

for i = 1:noconst
    result = [result, ', anal_names(i,:), ','...
        sprintf('%-10.3e', conc(i)), ', sprintf('%-10.3e', confid(i))];
end;

set(0, 'DiaryFile', diaryfile);
set(0, 'Diary', 'on');
fprintf(1,'Final result:%s\n',result);

set(0, 'Diary', 'off');
return;

```

### **dim\_set\_batch.m**

```

function [code] = dim_set_batch(dimenv)
% MATLAB function to setup the batch directories
% This file should be stored in the general control and communication dir.
%
% Written by: Leon N. Klatt
%           Robotics & Process Systems Division
%           Oak Ridge National Laboratory
%           P.O. Box 2008
%           Oak Ridge, TN 37831-6306
%           Initially created: February 28, 1995
%           Last Modified: April 3, 1996
%
% Modified by: Martin Hunt
%           Remove user interaction, create batch directory file
%           Last Modified: July 31, 1996
%
% Modified by: L. N. Klatt November 4, 1996 to make it agree with startup.m
%
% First define the installation specific paths for the original
% source code being used with the batch of samples for the GC DIM.
% The following group of lines will require definition for each
% installation of the GC DIM.
%
matlabcode = '/caa/dim/gc_dev';
nncode = '/caa/dim/CANNS';
generalcode = '/caa/dim/general';
software_code = '/software_code';
caagroup = 'caa';
%
% Start the set up process.
%
if(dimenv == []);
    fprintf(1, '%s\n', 'The variable containing the batch directory is not defined!');
    return;
else

```

```

elseif comb_mode == 10 % SLE raw
    fprintf(1,'Combination mode: SLE Raw\n');

% call method

[result, code] = sle_analyze(full_path_sample, plotting);
if code < 0
    return;
end;

%parse result

[name, conc, confid, import, code]= parse_result_string(result);
if code < 0
    return;
end;

%check to see which analyte reported

result = [];
for i = 1:noconst
    for j = 1:length(conc)
        if strcmp(anal_names(i,:), name(j,:))
            result = [result, ', ', anal_names(i,:), ',',...
                      sprintf('%-10.3e', conc(j), confid(j))];
        else
            result = [result, ', ', anal_names(i,:), ',',...
                      sprintf('%-10.3e', 0.0, -1.0)];
        end;
    end;
end;
result = [result, ', 'Importance = ', sprintf(' %7.3f, ...
import)];
result = fliplr(deblank(fliplr(deblank(result))));
result = [result, setstr(0)];

%parse complete result

[name, conc, confid, import, code]= parse_result_string(result);
if code < 0
    return;
end;

%
% write final result string
%
result = [];

```

```

elseif comb_mode == 5 % PCR peak
    fprintf(1,'Combination mode: PCR Peak\n');
% determine the *.peakset.fact file to use
    modelfile = [calset_prefix, '.peakset.fact'];

    [result, code] = dimpcrcal(modelfile, full_path_sample, plotting);
if code < 0
    return;
end;

[name, conc, confid, import, code]= parse_result_string(result);
if code < 0
    return;
end;

elseif comb_mode == 6 % MLR raw

elseif comb_mode == 7 % MLR peak

for i = 1:noconst
    [result, code] = runMLR(anal_names(i,:), full_path_sample, plotting);
    [name, tmp_conc, tmp_confid, import, code]= parse_result_string(result);
    conc(i)=tmp_conc;
    confid(i)=tmp_confid;
end
conc = conc';
confid = confid';

elseif comb_mode == 8 % Linear regression peak

elseif comb_mode == 9 % Neural network
    fprintf(1,'Combination mode: ANN Peak\n');
    network = [dim_batch_dir, '/nn/',calset_prefix,'.net'];

    [result, code] = ann_analysis(full_path_sample,network,calset_prefix);

if code < 0
    return;
end;

[name, conc, confid, import, code]= parse_result_string(result);
if code < 0
    return;
end;

```

```

elseif comb_mode == 3 % fusion
    fprintf(1,'Combination mode: Fusion\n');

    for i = 1:length(index)
        if comb_matrix(index(i),3) < 0,
            comb_matrix(index(i),3) = 0;
        elseif comb_matrix(index(i),3) > 1,
            comb_matrix(index(i),3) = 1;
        end;
    end;

    for i = 1:noconst

        index = find(~isnan(comb_matrix(:,((i-1)*2)+4)));
        %normalize weights to sum to one

        sum_weight = sum(comb_matrix(index,3));
        weight_vect = comb_matrix(index,3) / sum_weight;

        conc(i) = sum(comb_matrix(index,((i-1)*2)+4) .* weight_vect);
        % don't include methods with no confidence interval
        conf_index = find(comb_matrix(index,((i-1)*2)+5) ~= -1);
        sum_weight = sum(comb_matrix(index(conf_index),3));
        weight_vect = comb_matrix(index(conf_index),3) / sum_weight;
        confid(i) = sqrt(sum((comb_matrix(index(conf_index),((i-1)*2)+5).^2)...
            .* weight_vect));
    end;

    fprintf(1,'Summary of combination parameters:\n');
    comb_matrix

elseif comb_mode == 4 % PCR raw
    fprintf(1,'Combination mode: PCR Raw\n');
    % determine the *.calset.fact file to use
    modelfile = [calset_prefix, '.calset.fact'];

    [result, code] = dimpcrca(modelfile, full_path_sample, plotting);

    [name, conc, confid, import, code]= parse_result_string(result);
    if code < 0
        return;
    end;

```

```

else
    comb_matrix(row_index,3) = 1.0;
end

for i = 1:noconst
    comb_matrix(row_index,((i-1)*2)+4) = conc(i);
    comb_matrix(row_index,((i-1)*2)+5) = confid(i);
end;
%
% end SLE

set(0, 'DiaryFile', diaryfile);
set(0, 'Diary', 'on');

% combine results
%
index = find(comb_matrix(:,1));

if comb_mode == 0      % minimum
    fprintf(1,'Combination mode: Minimum\n');

    for i = 1:noconst
        valid_index = find(~isnan(comb_matrix(index,((i-1)*2)+4)));
        [conc(i), pick_index] = min(comb_matrix(valid_index,((i-1)*2)+4));
        confid(i) = comb_matrix(valid_index(pick_index),((i-1)*2)+5);
    end;

elseif comb_mode == 1 % maximum
    fprintf(1,'Combination mode: Maximum\n');

    for i = 1:noconst
        valid_index = find(~isnan(comb_matrix(index,((i-1)*2)+4)));
        [conc(i), pick_index] = max(comb_matrix(valid_index,((i-1)*2)+4));
        confid(i) = comb_matrix(valid_index(pick_index),((i-1)*2)+5);
    end;

elseif comb_mode == 2 % average
    fprintf(1,'Combination mode: Average\n');

    for i = 1:noconst
        valid_index = find(~isnan(comb_matrix(index,((i-1)*2)+4)));
        conc(i) = mean(comb_matrix(valid_index,((i-1)*2)+4));
    end;

% don't include methods with no confidence interval
    conf_index = find(comb_matrix(valid_index,((i-1)*2)+5) == -1);
    confid(i) = ...
        sqrt(mean(comb_matrix(valid_index(conf_index),((i-1)*2)+5).^2));
    end;

```

```

if code < 0
    return;
end;

%parse result

[name, conc, confid, import, code]= parse_result_string(result);
if code < 0
    return;
end;

%check to see which analyte reported

result = [];
for i = 1:noconst
    match_flag = 0;
    for j = 1:length(conc)
        if strcmp(anal_names(i,:), name(j,:))
            result = [result, ', anal_names(i,:), ', ...
                sprintf('%-10.3e', conc(j)), ',',...
                sprintf('%-10.3e', confid(j))];
            match_flag = 1;
        end;
    end;
    if match_flag == 0
        result = [result, ', anal_names(i,:), ',...
            sprintf('%-10.3e', 0.0, -1.0)];
    end;
end;

result = [result, ', 'Importance = ', sprintf(' %7.3f, ...
import)];
result = fliplr(deblank(fliplr(deblank(result))));
result = [result, setstr(0)];

% now parse result with all analytes

[name, conc, confid, import, code]= parse_result_string(result);

if code < 0
    return;
end;

comb_matrix(row_index,1) = 10;      %method is SLER
comb_matrix(row_index,2) = import;  %importance is column 2
if comb_mode == 3
    fis = readfis('sle_r');
    input = fis_range(['conc' import], fis);
    comb_matrix(row_index,3) = evalfis(input, fis);

```

```

    else
        comb_matrix(row_index+i-1,3) = 1.0;
    end
    comb_matrix(row_index+i-1,((i-1)*2)+4) = conc;
    comb_matrix(row_index+i-1,((i-1)*2)+5) = confid;

end

% end MLR Peak

row_index = row_index +3;

% Neural network
network = [dim_batch_dir, '/nn/', calset_prefix, '.net'];

[result, code] = ann_analysis(full_path_sample, network, calset_prefix);

if code < 0
    return;
end;

[name, conc, confid, import, code] = parse_result_string(result);
if code < 0
    return;
end;

comb_matrix(row_index,1) = 9;      %method is NNP
comb_matrix(row_index,2) = import; %importance is column 2
if comb_mode == 3
    fis = readfis('ann_p');
    input = fis_range(['conc' 'import'], fis);
    comb_matrix(row_index,3) = evalfis(input, fis);
else
    comb_matrix(row_index,3) = 1.0;
end

for i = 1:noconst
    comb_matrix(row_index,((i-1)*2)+4) = conc(i);
    comb_matrix(row_index,((i-1)*2)+5) = confid(i);
end;
%end ANN

row_index = row_index +1;

%SLE Raw

[result, code] = sle_analyze(full_path_sample, plotting);

```

```

for i = 1:noconst
    comb_matrix(row_index,((i-1)*2)+4) = conc(i);
    comb_matrix(row_index,((i-1)*2)+5) = confid(i);
end;
%
% end PCRR

row_index = row_index +1;

%PCR Peak
modelfile = [calset_prefix, '.peakset.fact'];
[result, code] = dimpercal(modelfile, full_path_sample, plotting);
[name, conc, confid, import, code]= parse_result_string(result);
if code < 0
    return;
end;

comb_matrix(row_index,1) = 5;      %method is PCRP
comb_matrix(row_index,2) = import; %importance is column 2
if comb_mode == 3
    fis = readfis('pcr_p');
    input = fis_range([conc' import], fis);
    comb_matrix(row_index,3) = evalfis(input, fis);
else
    comb_matrix(row_index,3) = 1.0;
end

for i = 1:noconst
    comb_matrix(row_index,((i-1)*2)+4) = conc(i);
    comb_matrix(row_index,((i-1)*2)+5) = confid(i);
end;

% end PCRP

row_index = row_index +1;

%MLR Peak
if comb_mode == 3
    fis = readfis('mlr_p');
end

for i = 1:noconst
    [result, code] = runMLR(anal_names(i,:), full_path_sample, plotting);
    [name, conc, confid, import, code]= parse_result_string(result);
    comb_matrix(row_index+i-1,1) = 7; %method is MLR Peak
    comb_matrix(row_index+i-1,2) = import;
    if comb_mode == 3
        input = fis_range([conc' import], fis);
        comb_matrix(row_index+i-1,3) = evalfis(input, fis);
    end
end

```

```

fprintf(1,'Check the contents of %s, improper number of analytes or standards .\n',...
    calset_file);
return;
end;

% read analyte name strings

for k = 1:noconst;
    dummyname = fscanf(batfile, '%s', 1);
    anal_names = str2mat(anal_names, dummyname);
end;

anal_names = anal_names(2:noconst+1,:);

fclose(batfile);

fprintf(1,'Analysis for the following analytes:\n');
for k = 1:noconst;
    fprintf(1,'#%d: %s\n',k, anal_names(k,:));
end;

% determine combination mode

if comb_mode >= 0 & comb_mode <= 3 % call all available methods
    comb_matrix = zeros(6+noconst, 3+2*noconst); %create matrix for results
    comb_matrix(:,2:3+2*noconst) = ones(6+noconst,2+2*noconst)*NaN;

    row_index = 1;
%PCR Raw
    modelfile = [calset_prefix, '.calset.fact'];
    [result, code] = dimpercal(modelfile, full_path_sample, plotting);
    if code < 0
        return;
    end;
    [name, conc, confid, import, code]= parse_result_string(result);
    if code < 0
        return;
    end;

    comb_matrix(row_index,1) = 4;      %method is PCRR
    comb_matrix(row_index,2) = import; %importance is column 2
    if comb_mode == 3
        fis = readfis('pcr_r');
        input = fis_range([conc import], fis);
        comb_matrix(row_index,3) = evalfis(input, fis);
    else
        comb_matrix(row_index,3) = 1.0;
    end

```

```

calset_file = fliplr(deblank(fliplr(deblank(calset_file))));

if length(calset_file) == 0
    if abs(calset_file(length(calset_file))) == 10 % new line
        calset_file = calset_file(1:length(calset_file)-1);
    end
else % error - could not find any *.calset files
    code = -470;
    fprintf(1,'Did not find any "*.calset" files in %s.\n',...
        [dim_batch_dir, '/standards/']);
    return;
end;

% extract the prefix of the *.calset file

slash_index = findstr(calset_file,'/');
ext_index = findstr(calset_file,'.calset');

if isempty(slash_index) | isempty(ext_index) |...
    slash_index(length(slash_index))+1 > (ext_index-1)
code = -472;
fprintf(1,'Could not isolate the prefix of the *.calset file:%s\n',...
    calset_file);
return;
end;

calset_prefix = calset_file(slash_index:length(slash_index)+1:(ext_index-1));

fprintf(1,'Using %s as the calibration set prefix\n', calset_prefix);

% determine the analyte names

batfile = fopen(calset_file, 'r');
if batfile == -1 % check for valid file
    code = -900;
    fprintf(1, 'File not found:%s\n',calset_file);
    return;
end;

% determine the number of standards

noconst = fscanf(batfile, '%g\n', 1);
nostand_str = fgetl(batfile);
nostand = sscanf(nostand_str, "%g",1);

% check the number of standards

if noconst < min_const | noconst > max_const | nostand < (noconst * min_stand)
    code = -272;

```

```

%           the current batch processing directory (as defined by the
%           return from "dim_getenv('DIMBATCH_DIR').
%
%   comb_mode  The type of combination of the results from the various methods
%               (might be just a single method). Numeric value based on
%               enumerated list defined in "general/dim_slm.h"

% initialize variables

code = 0;
result = [];
plotting = 0; %plotting flag as passed to methods
min_const = 1; %minimum number of analytes
max_const = 3; %maximum number of analytes
min_stand = 2; %minimum number of standards

dim_batch_dir = dim_getenv('DIMBATCH_DIR');
if isempty(dim_batch_dir)
    code = -473;
    fprintf(1, 'Batch directory environment variable not set\n');
    return;
end

%
% check for existence of sample
%
sample_dir = [sample_id,'.d'];
test = ls([dim_batch_dir,'/',sample_type,'s/',sample_dir]);
if isempty(test)
    code = -474;
    fprintf(1, 'Could not find unknown sample file:%s\n',...
        [dim_batch_dir,'/',sample_type,'s/',sample_dir]);
    return;
end;

full_path_sample = [dim_batch_dir,'/',sample_type,'s/',sample_dir,'',...
    sample_id,'.cdf.blr'];

%setup diary file
diaryfile = [dim_batch_dir,'/',sample_type,'s/',sample_dir,'/','dim_analyze.diary'];
set(0, 'DiaryFile', diaryfile);
set(0, 'Diary', 'on');
datetimestr = time_date(clock);
fprintf(1, '%s%s\n', 'Sample analyzed on: ', datetimestr);

% Read in *.calset file to determine the prefix.
%First find the file in the standards directory

calset_file = ls([dim_batch_dir, '/standards/*.calset']);

```

```

calset_prefix = calset_file(slash_index(length(slash_index))+1:(ext_index-1));

result = calset_prefix;

return;

```

### **dim\_analyze.m**

```

function [result, code]= dim_analyze(sample_type, sample_id, comb_mode);

% FILENAME dim_analyze.m
%
% MATLAB function which is called by the DIM control code (via the MATLAB
% engine) to compute the concentration of desired analytes in the unknown
% sample. The function will make the necessary calls to the individual
% methods based on the input arguments.
%
% Written by: Martin Hunt
%           Instrumentation and Controls Division
%           Oak Ridge National Laboratory
%           P.O. Box 2008, MS 6011
%           Oak Ridge, TN 37831-6011
% Initially created: 7/5/96
% Last Modified: 8/7/96 MAH
%
% Function call:
%
% [result, code]= dim_analyze(sample_type, sample_id, comb_mode);
%
% This function returns the following variables:
% result A string variable containing the results. The format of
% information within the string is:
% analyte_name analyte_conc conc_confid
% For multiple analytes this format is repeated for each analyte.
% For example, assume two analytes are included in the PCR
% calibration model, the returned string contains:
% "1242 6.75e+02 6.23e+1 1254 -2.33e-02 1.23e-01"
%
% code The integer execution code; zero indicates normal execution.
%
% The function requires the following input parameters:
% sample_type A string containing one of the valid AIA sample types
% (standard, unknown, control, blank) corresponding to the
% sample.
%
% sample_id A text string specifying the id of the sample to be analyzed.
% The file should be located in the "sample_type" subdirectory of

```

```

'if ac42 is absent and ac54 is absent and ac60 is absent and import is medium then
ann_p_weight is half ';
'if ac42 is absent and ac54 is absent and ac60 is absent and import is low then
ann_p_weight is zero  ];
ann_p_fis = parsrule(ann_p_fis, ruleTxt);
writefis(ann_p_fis,'ann_p');

```

### **cal\_prefix.m**

```

function [result, code] = cal_prefix();

dim_batch_dir = dim_getenv('DIMBATCH_DIR');

code = 0;

% Read in *.calset file to determine the prefix.
%First find the file in the standards directory

calset_file = ls([dim_batch_dir, '/standards/*.calset']);
calset_file = fliplr(deblank(fliplr(deblank(calset_file))));
if length(calset_file) == 0
    if abs(calset_file(length(calset_file))) == 10 % new line
        calset_file = calset_file(1:length(calset_file)-1);
    end
else % error - could not find any *.calset files
    code = -470;
    fprintf(1,'Did not find any "*.calset" files in %s.\n',...
        [dim_batch_dir, '/standards/']);
    return;
end;

% extract the prefix of the *.calset file

slash_index = findstr(calset_file,'/');
ext_index = findstr(calset_file,'.calset');

if isempty(slash_index) | isempty(ext_index) |...
    slash_index(length(slash_index))+1 > (ext_index-1)
    code = -472;
    fprintf(1,'Could not isolate the prefix of the *.calset file:%s\n',...
        calset_file);
    return;
end;

```

```

ann_p_fis = addmf(ann_p_fis, 'output',1,'zero','trimf',[-0.35 0 .35]);
ann_p_fis = addmf(ann_p_fis, 'output',1,'half','trimf',[0.15 0.5 0.85]);
ann_p_fis = addmf(ann_p_fis, 'output',1,'one','trimf',[0.65 1.0 1.35]);

ruleTxt=['if ac42 is present and ac54 is present and ac60 is present and import is high then
ann_p_weight is one ';
    'if ac42 is present and ac54 is present and ac60 is present and import is medium then
ann_p_weight is half';
    'if ac42 is present and ac54 is present and ac60 is present and import is low then
ann_p_weight is zero ';
    'if ac42 is present and ac54 is present and ac60 is absent and import is high then
ann_p_weight is one ';
    'if ac42 is present and ac54 is present and ac60 is absent and import is medium then
ann_p_weight is half';
    'if ac42 is present and ac54 is present and ac60 is absent and import is low then
ann_p_weight is zero ';
    'if ac42 is absent and ac54 is present and ac60 is present and import is high then
ann_p_weight is one ';
    'if ac42 is absent and ac54 is present and ac60 is present and import is medium then
ann_p_weight is half';
    'if ac42 is absent and ac54 is present and ac60 is present and import is low then
ann_p_weight is zero ';
    'if ac42 is present and ac54 is absent and ac60 is present and import is high then
ann_p_weight is one ';
    'if ac42 is present and ac54 is absent and ac60 is present and import is medium then
ann_p_weight is half';
    'if ac42 is present and ac54 is absent and ac60 is present and import is low then
ann_p_weight is zero ';
    'if ac42 is present and ac54 is absent and ac60 is absent and import is high then
ann_p_weight is half';
    'if ac42 is present and ac54 is absent and ac60 is absent and import is medium then
ann_p_weight is zero ';
    'if ac42 is present and ac54 is absent and ac60 is absent and import is low then
ann_p_weight is zero ';
    'if ac42 is absent and ac54 is present and ac60 is absent and import is high then
ann_p_weight is half';
    'if ac42 is absent and ac54 is present and ac60 is absent and import is medium then
ann_p_weight is zero ';
    'if ac42 is absent and ac54 is present and ac60 is absent and import is low then
ann_p_weight is zero ';
    'if ac42 is absent and ac54 is absent and ac60 is present and import is high then
ann_p_weight is half';
    'if ac42 is absent and ac54 is absent and ac60 is present and import is medium then
ann_p_weight is zero ';
    'if ac42 is absent and ac54 is absent and ac60 is present and import is low then
ann_p_weight is zero ';
    'if ac42 is absent and ac54 is absent and ac60 is absent and import is high then
ann_p_weight is one ';
    'if ac42 is absent and ac54 is absent and ac60 is absent and import is medium then
ann_p_weight is zero ';

```

## Appendix F: MATLAB function listings

### ann\_p\_fuzzy.m

```
% Generate a Fuzzy inference system
% For ANN method based on peak area input

% define the threshold for absence vs. presence for Aroclor concentration
abs_pres_thresh = 0.05;

%define the thresholds for importance - low, medium, high
imp_low_med = 0.15;
imp_med_high = 0.70;

% generate a Fuzzy inference system
ann_p_fis = newfis('ann_p','mamdani');

% add input variables
%Aroclor 1242
ann_p_fis = addvar(ann_p_fis,'input','ac42',[0 2]);

ann_p_fis = addmf(ann_p_fis, 'input',1,'absent','gauss2mf',[0.02378 0 0.03 0.02]);
ann_p_fis = addmf(ann_p_fis, 'input',1,'present','gauss2mf',[0.03 0.1 0.6795 2.2]);

%Aroclor 1254
ann_p_fis = addvar(ann_p_fis,'input','ac54',[0 2]);

ann_p_fis = addmf(ann_p_fis, 'input',2,'absent','gauss2mf',[0.02378 0 0.03 0.02]);
ann_p_fis = addmf(ann_p_fis, 'input',2,'present','gauss2mf',[0.03 0.1 0.6795 2.2]);

%Aroclor 1260
ann_p_fis = addvar(ann_p_fis,'input','ac60',[0 2]);

ann_p_fis = addmf(ann_p_fis, 'input',3,'absent','gauss2mf',[0.02378 0 0.03 0.02]);
ann_p_fis = addmf(ann_p_fis, 'input',3,'present','gauss2mf',[0.03 0.1 0.6795 2.2]);

%Importance
ann_p_fis = addvar(ann_p_fis,'input','import',[-1 1]);

ann_p_fis = addmf(ann_p_fis, 'input',4,'low','gauss2mf',[0.34 -1.1 0.1 0.0]);
ann_p_fis = addmf(ann_p_fis, 'input',4,'medium','gauss2mf',[0.2 0.4 0.2 0.5]);
ann_p_fis = addmf(ann_p_fis, 'input',4,'high','gauss2mf',[0.1 0.9 0.34 1.1]);

%Output variables

ann_p_fis = addvar(ann_p_fis,'output','ann_p_weight',[-0.4 1.4]);
```

```

#      rm -f *.o *.v *.V *.a

# This prints out source files that have been changed

notebook: $(SRCS) $(HDRS) $(RPCX)
        $(TOOLKIT)/mknotebook $?
        touch notebook

# This should be almost everywhere that I care about that include files are
# to be found.

CTAG_INCLUDES= \
$(VW_HOME)/h/* $(VW_HOME)/h/*/* \
*.h \
$(INCL:SCCS_HOME%=$(USR_HOME))/*.h \
$(INCL:SCCS_HOME%=$(SCCS_HOME))/*.h

CTAGS= /u2/gcc/bin/ctags -T -S -a -d

# This is real handy if you are using VI
tags: $(SRCS.c) $(SRCS.C)
      rm -f tags
      $(CTAGS) $(SRCS.C) $(SRCS.c) $(CTAG_INCLUDES) $(RPCX:.x=.h)
      mv tags raw_tags
      sort raw_tags>tags
      rm -f raw_tags

# It is a good idea to have Makefile checked out before doing this!
depend:
        $(TOOLKIT)/makedepend $(CPPFLAGS) $(USRCS)
        $(TOOLKIT)/makedepend -o.v -s"# VXWORKS DEPENDS:" $(VCPPFLAGS)
$ (VSRCS)

```

```

# Build the unix executable file
# Indicate your libraries and their locations here
$(TARGET): $(UOBJS)
    $(ULINK.C) $(ULIBS) $(UOBJS) $(MOTIF_LIBS) -o $@

servertest: $(UOBJS) servertest.o
    $(ULINK.C) servertest.o $(ULIBS) $(UOBJS) $(MOTIF_LIBS) -o $@

dim_slm: $(UOBJS)
    $(ULINK.C) $(UOBJS) $(ULIBS) -o $@

msg_que: msg_que_1 matlab_eng

msg_que_1: msg_que_1.c
    $(ULINK.C) msg_que_1.c $(ULIBS) -o $@

matlab_eng: matlab_eng.c
    $(ULINK.C) matlab_eng.c $(ULIBS) -o $@

# This is the centerline load target
# This rule should be very similar to the $(TARGET) rule above
# From the codecenter/objectcenter type "make centerline"
centerline: $(UOBJS)
    # load $(CFLAGS) $(CPPFLAGS) $(UOBJS) $(MOTIF_LIBS)

ULIBOBJJS= $(ULIBSRCS.C:.c=.o) $(ULIBSRCS.c:.c=.o) \
           $(CLIBSRCS.C:.c=.o) $(CLIBSRCS.c:.c=.o) \
           $(RPCX.o)

# Unix library construction add this to the all: rule if you want it
lib$(TARGET).a: $(ULIBOBJJS)
    @echo Adding to library $@ ; $?
    $(AR) $(ARFLAGS) $@ $?
    ranlib $@


***** Common goodies *****

SRCS=$(USRCS.C) $(USRCS.c) $(VSRCS.C) $(VSRCS.c) $(CSRCS.C) $(CSRCS.c) \
      $(ULIBSRCS.C) $(ULIBSRCS.c) $(VLIBSRCS.C) $(VLIBSRCS.c) \
      $(CLIBSRCS.C) $(CLIBSRCS.c)

# This removes all files that are not source files
clean:
#     rm -f $(RPCX:.x=.h) $(RPCX.c)
#     rm -f $(TARGETS) $(UOBJS) $(VOBJS) $(ULIBOBJJS) $(VLIBOBJJS)

```

```

VCFLAGS += -g

# vxworks Library construction

VLIBOBJJS= $(VLIBSRCS.C:.C=.v) $(VLIBSRCS.c:.c=.v) \
$(CLIBSRCS.C:.C=.v) $(CLIBSRCS.c:.c=.v) \
$(RPCX.v)

lib$(TARGET)_v.a: $(VLIBOBJJS)
    @echo Adding to library $@ : $?
    $(AR68) $(ARFLAGS) $@ $?
    ranlib $@

# Use the following for linking vxWorks files.
# VLINK.C is for c++ with global objects and VLINK.c is for all else.
# TARGET is defined as the name of this dir
# Use VLINK.c if there are no global C++ objects or if this file will
# be linked again.

$(TARGET).V: $(VOBJJS)
    $(VLINK.c) -o $@ $(VOBJJS)
#    cp $@ /u0/$USER

***** \
#*          Unix Support rules          *
***** /

```

CPPFLAGS += -I. -I.. -Icaa/tsc/cm/include -I/opt/matlab/extern/include

CFLAGS += -g -DDEBUG  
CFLAGS += -I/opt/netcdf/libsrc -I/opt/netcdf/xdr  
CFLAGS += -Lcaa/tsc/cm/lib/sparcSol2 -L/opt/matlab/extern/lib/sol2

# UNIX objects

UOJJS=\$(USRCS.C:.C=.o) \$(USRCS.c:.c=.o) \
\$(CSRCS.C:.C=.o) \$(CSRCS.c:.c=.o) \
\$(RPCX.o) \
/caa/tsc/cm/utils/src/getstdin.o

USRCS= \$(USRCS.C) \$(USRCS.c) \$(CSRCS.C) \$(CSRCS.c) \
\$(ULIBSRCS.C) \$(ULIBSRCS.c) \$(CLIBSRCS.C) \$(CLIBSRCS.c) \

```

all: $(TARGETS)

# Fill in LIBS with a list of linked in library files
# ..../libxxx/libusr.a - This rule will find and make the lib when needed
LIBS =
$(LIBS) : FORCE_LIBS
    cd $($@D) ; $(MAKE) $($@F)
FORCE_LIBS:

*****/*
#
#          RPC Support
*****/

# This makes sure that rpcgen headers get rebuilt - add any other targets
# that make might not be able to infer.
.INIT: $(RPCX:.x=.h) $(HDRS)

# Next lines needed because of rpcgen.new
# Enable this if you want to run the new rpcgen
RPCGEN=$(TOOLKIT)/rpcgen.new -C -N -b
# No main in server
RPC_SVC= -m

# rpc targets
RPCX.c= $(RPCX:.x=_xdr.c) $(RPCX:.x=_svc.c) $(RPCX:.x=_clnt.c)
RPCX.v= $(RPCX.c:.c=.v)
RPCX.o= $(RPCX.c:.c=.o)

# This allows make to complete the inferences for making rpc files
$(RPCX:.x=.h) $(RPCX:.x=_xdr.c) $(RPCX:.x=_svc.c) $(RPCX:.x=_clnt.c): $(RPCX)

*****\
#*
#          VxWorks Support rules
#*
*****/

```

#VXWorks objects

```

VOBJS=$(VSRCS.C:.C=.v) $(VSRCS.c:.c=.v) \
      $(CSRCS.C:.C=.v) $(CSRCS.c:.c=.v)
VSRCS= $(VSRCS.C) $(VSRCS.c) $(CSRCS.C) $(CSRCS.c) \
      $(VLIBSRCS.C) $(VLIBSRCS.c) $(CLIBSRCS.C) $(CLIBSRCS.c)

VCPPFLAGS += -g -I. -I.. $(INCL:SCCS_HOME%=-I$(USR_HOME)%)\ 
                     $(INCL:SCCS_HOME%=-I$(SCCS_HOME)%)
# Add this to VCPPFLAGS to fix some weird stuff
#     -D __GNUC__ _TYPEOF FEATURE_BROKEN_USE_DEFAULT_UNTIL_FIXED_

```

```

# Any special header files -
HDRS=
#TARGET=dim_slm
# Files should appear only once in these lists
# Common to both unix and vw
CSRCS.C=
CSRCS.c=
CLIBSRCS.C=
CLIBSRCS.c=

# Unix only sources
USRCS.C=err_code.C
USRCS.c=dim_slm.c \
    aia_poke.c \
    dim_util.c
ULIBSRCS.C=
ULIBSRCS.c=dim_util.c
ULIBS=-lmat -lm -lssl -lsocket -lcmc

# Vx only sources
VSRCS.C=
VSRCS.c=
VLIBSRCS.C=
VLIBSRCS.c=

# RPCGEN input files
RPCX=


# Point this at the root of your project SCCS tree
SCCS_HOME=/u2/projects
# Point this at the corresponding place in your local tree
USR_HOME=../..
```

# Where to search for include files, a list of directories relative to  
# SCCS\_HOME except that SCCS\_HOME/ is the top of each directory name  
# i.e. SCCS\_HOME/myproj/include SCCS\_HOME/mylib/h

INCL=

```

#TARGET:sh= basename `pwd`

# Build the unix target and the vxworks target
# Select your real target from below:
# Key:
#   unix lib   unix exe   vx lib   vx load and exe
TARGETS= lib$(TARGET).a $(TARGET) lib$(TARGET)_v.a $(TARGET).V

# This is what gets built
```

## Appendix E: Makefile

```
# %Q%
#*****#
#          Oak Ridge National Laboratory
#          Robotics and Process Systems Division
#          U. S. Government internal use only
#
# FILE: Makefile
# PROJECT: RES/RCS System
# AUTHOR: David H. Thompson
# VERSION: %I%
# Modified: %G% %U%
#
# Description:
#
#
# HISTORY:
# #   Date      By           Comment
# -----
# 1.0  11/09/95  DHT  Initial file creation.
#*****#
# SCCS id = %W%

# Instructions:
# USRCS are unix sources
# VSRCS are vxworks sources
# CSRCS are common to both os
# HDRS are all header files

MAKE_TOOLS=/rpsd/u2/USER_CONTRIB/make_tools
TOOLKIT=/rpsd/u2/USER_CONTRIB/daves-toolkit

#include for vxWorks projects
#include $(MAKE_TOOLS)/vw-rpc.mk
include $(MAKE_TOOLS)/cc-gcc-2.7.0.mk

#
# The following line enables the ansi c compiler for .c files
#CC=clcc

# Choose which X that you want - openwin is really X11R4 - x11 is openwin
#include $(MAKE_TOOLS)/x11.mk
#include $(MAKE_TOOLS)/x11r5.mk

# --- Fill in below:
```

```

}

rtrn = msgsnd(tmsqid, tmsgp, msgsz, msgflg);
if (rtrn == -1) {
    perror("msgop: msgsnd failed");
    break;
}

}

if(rtrn < 0) { /* error condition exit */
    fprintf(stderr,"%s exiting because of system call or MATLAB engine error\n", argv[0]);
} else {
    fprintf(stderr,"%s exiting because of operator command\n",argv[0]);
}

/* free memory for MATLAB output buffer */
free(mat_output);

/* free memory for message queues */
free(rmsgp);
free(tmmsgp);

/*close MATLAB engine */
engClose(ep);

}

```

```

ret_typ = 2;
tmsgp->mtype = ret_typ;
}else {
    fprintf(stderr, "MATLAB engine success\n");
    ret_typ = 1;
    tmsgp->mtype = ret_typ;
    data = shmat(shmid, (char *)0 /* let system determine */, 0);
    if(data == (char *)-1) {
        perror("shmat: shmat failed");
        break;
    }
    sprintf(stderr,"%s",mat_output);
    memccpy(data, mat_output, '\0', mat_out_size);
    if(engGetFull(ep, "code", &m, &n, &Dreal, &Dimg)==0) {
        return_code = (int) Dreal[0];
        fprintf(stderr,"Return code from MATLAB:%d\n",return_code);
        tmsgp->code = return_code;
    } else {
        return_code = 1;
        fprintf(stderr,"Return code not set in MATLAB, default:%d\n",return_code);
        tmsgp->code = -1;
    }
}

if(engGetFull(ep, "result", &m, &n, &Dreal, &Dimg)==0) {

    i=0;
    fprintf(stderr,"In %s, converting MATLAB double to character\n",argv[0]);
    if(Dreal != (double *)NULL) {
        fprintf(stderr,"In %s, size of \"result\" %d x %d %f\n", argv[0],m, n, Dreal[0]);
        while((tmp_char=(char)Dreal[i])!='\0' && i<(MSG_SIZE-1) && i<m*n) {
            fprintf(stderr,"%c", tmp_char);
            tmsgp->result[i++] = tmp_char;
        }
        tmsgp->result[i] = '\0';
    }
    /* msgsz = i+1; */
    fprintf(stderr,"%s: text sent:%s\n",argv[0], tmsgp->result);
} else {
    fprintf(stderr,"%s: variable \"result\" is empty\n",argv[0]);
}
} else {
    strcpy(tmsgp->result, "Could not open variable result\n");
}

rtrn = shmdt(data);
if (rtrn == -1) {
    perror("shmop: shmdt failed");
    break;
}

```

```

    }

/* */
/* access shared memory segment           */
/* */

if ((shmid = shmget(rkey, 20000, 0)) == -1) {
    perror("shmget: shmget failed");
    exit(1);
}

/* */
/* start MATLAB engine */
/* */

sprintf(matlab_strt, "\0");
if (!(ep = engOpen( matlab_strt ))) {
    fprintf(stderr, "\nCan't start MATLAB engine\n");
    exit(1);
}

mat_output = (char *) malloc(mat_out_size);
if (mat_output == (char *) NULL) {
    fprintf(stderr, "Could not allocate output buffer for MATLAB engine.\n");
    fprintf(stderr, "No output from MATLAB engine will be displayed.\n");
    mat_out_size = 0;
}

/* enable the output from MATALB engine to be captured and displayed to */
/* screen                         */
engOutputBuffer(ep, mat_output, mat_out_size);

while (1) {

rtrn = msgrcv(rmsqid, rmsgp, msgsz, msgtyp, msgflg);
if (rtrn == -1) {
    perror("msgop: msgrcv failed");
    break;
}

if (strcmp(rmsgp->command, "exit") == 0) break;

fprintf(stderr, "%s\n", rmsgp->command);

/* send command to MATLAB */

if (engEvalString(ep, rmsgp->command) != 0) {
    fprintf(stderr, "MATLAB engine error\n");
}

```

```

sleep(4);
fprintf(stdout, "In matlab_eng\n");

/* */
/* generate keys based on file and code          */
/* */
rkey = ftok("/usr", 'A');
tkey = ftok("/usr", 'B');

if(rkey == -1 || tkey == -1) {
    fprintf(stderr,"ftok failure in %s, exiting\n",argv[0]);
    exit(1);
}

/* */
/* allocate memory for message structure          */
/* */
rmsgp = (Cmd_msg_bufP) malloc((unsigned) sizeof(Cmd_msg_buf));
if (rmsgp == NULL) {
    (void) fprintf (stderr, "msgop: %s %d byte messages\n",
                   "could not allocate message buffer for", MSG_SIZE);
    exit(1);
}
tmsgp = (Rtn_msg_bufP) malloc((unsigned) sizeof(Rtn_msg_buf));
if (tmsgp == NULL) {
    (void) fprintf (stderr, "msgop: %s %d byte messages\n",
                   "could not allocate message buffer for", MSG_SIZE);
    exit(1);
}

/* */
/* set flags and types */
/* */
tmsgp->mtype = 1;
msgtyp = 0L;
msgflg = 0;
msgsiz = MSG_SIZE;

/* */
/* access the message channel          */
/* */
if((rmsqid = msgget(rkey, 0)) == -1)
{ perror("msgget: msgget failed");
  exit(1);
}
if((tmsqid = msgget(tkey, 0666)) == -1)
{ perror("msgget: msgget failed");
  exit(1);
}

```

```

listem:
while ( (ptr->errnum != -9998) ) {
    printf("%d %s\n",ptr->errnum, ptr->funcname);
    printf("%s\n\n",ptr->verbose);
    ptr++;
}
ptr_ret=(ptr->funcname);
return (ptr_ret);

} // end of decipher_error_codes

matlab_eng.c
/*
 * process to execute MATLAB functions */
/*
#include      <stdio.h>
#include      <sys/types.h>
#include      <sys/ipc.h>
#include      <sys/msg.h>
#include "dim_slm.h"

int main(int argc, char *argv[])
{
    key_t      tkey; /* unique key for send message queue */
    key_t      rkey; /* unique key for receive message queue */
    int        tmsqid; /* returned send message queue id */
    int        rmsqid; /* returned receive message queue id */
    int        shmid; /* returned shared memory id */
    Rtn_msg_bufP tmsgp; /* message send structure */
    Cmd_msg_bufP rmsgp; /* receive structure */
    int        msgsz, msgflg;
    long       msgtyp, ret_typ;
    int        i, j;
    int        rtrn, return_code;
    FILE      *fd;
    char      *data; /* pointer to shared memory area */
    char      *textp, tmp_char;

    Engine *ep;
    char *mat_output;
    char matlab_strt[LBUFF_SIZE];
    int mat_out_size = MATLAB_OUTPUT_SIZE;
    double *code, *Dreal, *Dimg;
    int m, n;

/* wait until other process has opened IPC stuff */

```

```

{"analurr.m",      -471, "\tThe UNIX \"basename\" command failed.\n"},

 {"analurr.m",      -472, "\tThe UNIX \"dirname\" command failed.\n"},

 {"analurr.m",      -473, "\tThe sample directory does not exist.\n"},

 {"analurr.m",      -474, "\tThe surrogate calibration model file could not\n"
 " \tbe opened."},

 {"analurr.m",      -475, "\tThe identified model file is not a SUR\n"
 " \tcalibraion model file."},

 {"runSUR.m",       -480, "\tThe surrogate model filename passed to the function\n"
 " \tstarts at the root directory; this is not allowed."},

 {"runSUR.m",       -481, "\tThe specified surrogate model filename does not exist.\n"},

 {"bldpcrpeaktime.m", -520, "\tThe UNIX command \"ls\" failed, probably means no\n"
 " \t*.peakset files were found."},

 {"bldpcrpeaktime.m", -521, "\tThe user did not select a *.peakset file.\n"},

 {"bldmlrpeaktime.m", -530, "\tThe UNIX command \"ls\" failed, probably means no\n"
 " \t*.peakset files were found."},

 {"bldmlrpeaktime.m", -531, "\tThe user did not select a *.peakset file.\n"},

 {"decipher_error_codes", -9998, "\tValue of \"iopt\" is not valid."},

 {"decipher_error_codes", -9999, "\tThe error code given is not valid or is undefined."} };

/* Additional error codes may be added to the above structure */
/* provided that codes -9998 and -9999, and their associates, */
/* are the last two entries. */

struct errcodes *ptr = &described[0];
char *ptr_ret;

if (iopt == 9) goto listem;
if (iopt != 1 && iopt !=2) {numerr = -9998 ;}
while( (ptr->errnum != numerr) ) {
    ptr++;
    if (ptr->errnum == -9999) break;
}

if(iopt == 1) {ptr_ret=(ptr->funcname); }
if(iopt == 2) {ptr_ret=(ptr->verbose); }
return (ptr_ret);

```

```

{"readPEAKset.m",      -413, "\tAn unsupported model type was specified."},  

{"readPEAKset.m",      -414, "\tThe UNIX function to obtain the basename of the\n"
                           "\tfile failed."},  

{"readPEAKset.m",      -415, "\tA *.peakset file was not identified."},  

{"readPEAKset.m",      -416, "\tThe peak time parameter file for the specified NN\n"
                           "\tcalibration model does not exist."},  

{"readPEAKset.m",      -417, "\tThe retention time marker parameter file could not\n"
                           "\topened."},  

{"readPEAKset.m",      -418, "\tThe peak time parameter file for the NN calibration\n"
                           "\tmodel could not be opened."},  

{"readPEAKset.m",      -419, "\tThe peak time parameter file for the specified\n"
                           "\tsurrogate calibration model does not exist."},  

{"peaknorm.m",         -420, "\tCould not open file to store the peak-area normalized\n"
                           "\tdata."},  

{"dim_setenv.m",        -430, "\tAn illegal environment variable was passed to the\n"
                           "\tfunction."},  

{"dim_getenv.m",        -440, "\tAn illegal environment variable was passed to the\n"
                           "\tfunction."},  

{"choosepeaks.m",       -450, "\tThe retention time parameter file could not be\n"
                           "\topened."},  

{"choosepeaks.m",       -451, "\tThe specified *.peakset file could not be opened."},  

{"surrmodbld.m",        -460, "\tNo surrogate calibration data files are present.\n"},  

{"surrmodbld.m",        -461, "\tThe UNIX \"basename\" command failed.\n"},  

{"surrmodbld.m",        -462, "\tA surrogate calibration data file was not selected.\n"},  

{"surrmodbld.m",        -463, "\tThe task of building a new surrogate calbraion\n"
                           "\tmodel was not completed."},  

{"surrmodbld.m",        -464, "\tThe peak time file for the selected surrogate could\n"
                           "\tnot be located."},  

{"analssurr.m",         -470, "\tA *.peakset.fact calibration model file for the\n"
                           "\tsurrogate could not be located."},
```

```

{"qa_cal_check.m", -365, "\tThe analyte name in the result string does not agree\n"
"\twith the name in the \"sample_name\" attribute of the\n"
"\tdata file header."},

 {"qa_cal_check.m", -366, "\tDuplicate analyte names were found in the result\n"
"\tstring."},

 {"qa_cal_check.m", -367, "\tThe calibration check failed."},

 {"parse_result_string.m", -370, "\tThe result array is undefined."},

 {"parse_result_string.m", -371, "\tThe result array is not a string array."},

 {"readGCPeak.m", -380, "\tError from \"mexcdf\" while attempting to ascertain the\n"
"\tnumber of peaks in the AIA formatted data file."},

 {"readGCPeak.m", -381, "\tError from \"mexcdf\" while attempting to read the\n"
"\ttretention times."},

 {"readGCPeak.m", -382, "\tError from \"mexcdf\" while attempting to read the\n"
"\tpeak areas."},

 {"readGCPeak.m", -383, "\tError from \"mexcdf\" while attempting to read the\n"
"\tpeak heights."},

 {"readGCPeak.m", -384, "\tError from \"mexcdf\" while attempting to read the\n"
"\tpeak widths."},

 {"readPeakdata.m", -390, "\tThe peak time parameter file for the specified PCR\n"
"\tcalibration model does not exist."},

 {"readPeakdata.m", -391, "\tThe peak time parameter file for the specified MLR\n"
"\tcalibration model does not exist."},

 {"readPeakdata.m", -392, "\tAn unsupported model type was specified."},

 {"readPeakdata.m", -393, "\tThe UNIX function to obtain the directory name\n"
"\tfrom a filename failed."},

 {"readAIAPEAK.m", -400, "\tOpening of the specified retention time parameter file\n"
"\tfailed."},

 {"readPEAKset.m", -410, "\tThe specified calibration file could not be located."},

 {"readPEAKset.m", -411, "\tThe peak time parameter file for the specified PCR\n"
"\tcalibration model does not exist."},

 {"readPEAKset.m", -412, "\tThe peak time parameter file for the specified MLR\n"
"\tcalibration model does not exist."},

```

```
{"qa_retent_mark.m", -331, "\tParameter file could not be opened for reading."},  
 {"qa_retent_mark.m", -332, "\tParameter file could not be opened for writing."},  
 {"qa_retent_mark.m", -333, "\tUNIX command to obtain the directory name for the\n" "\tdata file failed."},  
 {"qa_retent_mark.m", -334, "\tFile \"retentionmarkers.dat\" could not be opened for\n" "\treading."},  
 {"qa_retent_mark.m", -335, "\tData inconsistency in file \"retentionmarkers.dat\"."},  
 {"qa_retent_mark.m", -336, "\tSample data file could not be located in the currently\n" "\tdefined data subdirectory."},  
 {"qa_retent_mark.m", -337, "\tRetention time marker(s) failed the QA evaluation."},  
 {"qa_off_scale.m", -340, "\tIncorrect number of parameters passed to the\n" "\tfunction."},  
 {"qa_off_scale.m", -341, "\tSample data file could not be located in the currently\n" "\tdefined data subdirectory."},  
 {"qa_off_scale.m", -342, "\tUNIX command to obtain the directory name for the\n" "\tdata file failed."},  
 {"qa_off_scale.m", -343, "\tFound ordinate values that are either at the detector\n" "\tminimum or detector maximum, hence, the data is\n" "\tconsidered off-scale."},  
 {"gc_isolate_peak.m", -350, "\tNo data available."},  
 {"gc_isolate_peak.m", -351, "\tNo retention time specified for the peak location."},  
 {"gc_isolate_peak.m", -352, "\tThe argument count is incorrect."},  
 {"qa_cal_check.m", -360, "\tSample data file could not be located in the currently\n" "\tdefined data subdirectory."},  
 {"qa_cal_check.m", -361, "\tUNIX command to obtain the directory name for the\n" "\tdata file failed."},  
 {"qa_cal_check.m", -362, "\tThe reading of attributes from the data file failed."},  
 {"qa_cal_check.m", -363, "\tThe sample_amount variable is not defined."},  
 {"qa_cal_check.m", -364, "\tThe sample_amount is a string variable; it must be a\n" "\tfloating point variable."},
```