MDI: *Mathematica* Database Interface for the MFEDB

J. C. Wiley, W. H. Miner, Jr. and D. W. Ross

DOE/ER/53266--39

DE92 014337

Fusion Research Center
The University of Texas at Austin
Austin, Texas 78712

April 1992

Abstract

We describe a new interface for the Magnetic Fusion Energy Database, MFEDB, which uses *Mathematica*® as a front end. MDI is a *Mathematica* package that defines a basic set of MFEDB access functions. The package will also accept standard SQL queries. Each function returns *Mathematica*-style lists, which can then be manipulated with any of the *Mathematica* functions. MDI also provides some utility functions for plotting and analyzing the data. The MDI package essentially makes the MFEDB an extension of *Mathematica*. The user may use any of the many *Mathematica* front-ends including telnet, X-Windows, or a notebook. The mdi.m package may be obtained by anonymous FTP from the MFEDB site or by use of netmfe, an E-mail database interface.

MDI is a example of distributed computing. Behind the user interface, MDI calls an RPC client program that communicates with an RPC server on the MFEDB computer. It relies on the network communication capabilities of *Mathematica* to connect the user to a workstation running the *Mathematica* kernel. The *Mathematica* kernel is then connected to the MFEDB host workstation by a client/server pair of RPC processes. If the *Mathematica* kernel is to be run on the user's machine, the RPC client program must also be obtained and installed.

The MDI RPC server is also available for users who would like to provide their own client software. The server returns ASCII tables from standard queries and may be accessed and processed by any program on the internet that has access to RPC services.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MASTER

I. Introduction

This report serves two functions. It provides an introduction and user's manual to MDI, and it describes in detail the underlying software, which may provide a useful example for others developing similar distributed applications. The MDI package is useful in its own right as it allows users to apply the powerful set of *Mathematica*[0] operations directly to the data in the MFEDB[1] database. The combination of MDI and *Mathematica* allows users to view, extract, and manipulate the MFEDB without having to understand either the SQL[2] query language or some of the more subtle interrelationships of the underlying relational database. The MDI package does allow more experienced users to present any SQL query to the database from within a *Mathematica* interface.

This report also describes the MDI RPC[3] daemon, which provides network access to the MFEDB. Users may develop their own client programs to extract data from the database without using either the MDI client program, *MDIClient,* or the *Mathematica* interface. In this case the *MDIClient* program serves as an example.

Why MDI? Prior to the development of MDI, there were two methods of accessing the MFEDB. The most direct method required that the user obtain an account on the MFEDB host computer and use Oracle's SQL*PLUS®[4] query program. Once data had been extracted into a local file it was then usually sent by FTP[5] to the user's host machine for further processing. This method is suitable for those users with good network access to the MFEDB host machine and who are familiar with the SQL query language. It is not particularly convenient for a cursory examination of the data or for casual users.

A second access method was modeled after the *netlib*[6] software. *Netlib* is a program that accepts E-mail requests for numerical-analysis software or other information about the numerical-analysis community and sends back by return E-mail an answer, which may be a program or other text file. The *netmfedb*[7] service operates in a similar manner. It accepts a fixed set of requests by E-mail and responds by return E-mail. The commands accepted are of three types: requests for particular files, requests to execute predefined queries, and requests to execute a general SQL query. The *netmfedb* program solved three problems with MFEDB access. Firstly, the user no longer needed to have an account on the host machine. The *netmfedb* accepts requests from anyone who has access to Internet E-mail. Secondly, it expanded the potential users to those who have E-mail but not high-quality network access to the MFEDB host. Finally, the netmfedb program allowed certain queries to be packaged so that the user can request information without having to generate an SQL query or understand in detail the interrelationships between tables.

MDI provides additional capabilities beyond those provided by either SQL*PLUS or *netmfedb* and has different network access requirements. The MDI *Mathematica* package

adds commands to *Mathematica* to access the MFEDB and to transfer and manipulate MFEDB data within a *Mathematica* environment. The powerful *Mathematica* list-processing, computational, and graphics capabilities can be directly applied to the MFEDB. As part of the MDI system, an RPC service has been established on the MFEDB host computer. This service provides MFEDB access to programs that have RPC network access but do not require *Mathematica.*

In order to use the *Mathematica* package, the user obviously must have access to *Mathematica.* There are two ways this can be accomplished. Users with a *Mathematica* front end installed can simply use the *Mathematica* network connection tools to connect to the *Mathematica* kernel running on a workstation at the Fusion Research Center. A special account has been set up for this purpose.

Users who would like to run the *Mathematica* kernel on their own machines or to access the MFEDB from within their own programs have somewhat different requirements. The user's host machine must have RPC access to the MFEDB host machine, but the user is not required to obtain an account on the MFEDB host. The MDI user must also be able to install either the *MDIClient* or a similar RPC client program on his host computer. For *MDIClient*, this requires a C compiler and an RPC library. These should be available on most Unix machines and on VMS machines equipped with Multinet® or equivalent TCP/IP[8] software.

The remainder of this report is organized as follows: Section II describes the *Mathematica* commands that MDI defines and serves as a user's guide to MDI. Section III presents some examples of how the MDI commands can be combined with standard *Mathematica* commands to generate more powerful queries and reports. It is the combination of Mathematica and MFEDB that allows very powerful data access and manipulation commands to be constructed. Section IV describes the client/server RPC model and describes the *Mathematica-MDIClient* interaction. Section V describes the MDI RPC server. Appendix I contains a diagram of the MFEDB. Appendix II is a listing of the *MDIClient* code and Appendix III is a listing of the MDI RPC server software. The MDI package is listed in Appendix IV.

II. MDI commands

MDI is a *Mathematica* package, mdi.m, which must first be loaded into the *Mathematica* environment. For users running the kernel on the Fusion Research Center workstation, this can be accomplished with the command `<<Local`mdi`.` Each of the MDI commands begins with the prefix "Mdi". A complete list of commands can be found after loading the MDI package by executing the *Mathematica* command `?Mdi*`. A brief

description of each command may be found by using the *Mathematica* ? operator. Most MDI commands return either a list of strings or a graphic object. The strings may need to be converted into numbers for certain *Mathematica* operations. The function `MdiToNumber` is provided for this operation. The `MdiProfile` and `MdiProfileXY` commands convert the lists into numbers directly.

A. MFEDB

To make use of the MFEDB one should be familiar with some of the terms used in the database and the general organizational structure of the database. Appendix I contains a diagram of the internal relational structure of the MFEDB and shows the important links between the tables. Use of the MDI command largely obviates the need to understand how these links are used or the organization of the database; however the MFEDB does define certain terminology that should be understood. Each experiment is referred to as a machine. For each machine the data is organized in terms of discharges or shots and these discharges are numbered with a shot number. Experimentalists often refer to particular discharges in terms of certain nominal parameters, even though these parameters may only be approximately obtained during part of the discharge. These parameters describe the discharge conditions and include the toroidal field, plasma current, average electron density, etc. Discharges are also often characterized by different terms such as the type of heating, ohmic, beam, etc., or the boundary conditions, limited, diverted; these terms are called keywords. Keywords are words that describe the discharge without needing an associated value. Other scalar characteristics of the discharge, such as particle confinement time require both a parameter name and value. These are called characteristics, and the value is the characteristic value.

The MFEDB is not an experimental archive and does not store raw data. All data has been analyzed and converted to physically meaningful quantities, such as electron temperature, electron density, etc. These physical quantities are usually measured at multiple points across a discharge at a given time to form a radial profile or are measured at a given position for multiple points in time to form a temporal profile. In general, quantities that have values associated with an independent variable are called profiles. Each profile is characterized by its name, the independent variable, either space or time, the coordinate system, how the data are stored etc. Each profile is stored either as a fit to a specified function or as data-point four-tuples, (x, dx, y ,dy), although the error values are often zero. Spatial profiles that have been taken at several times during a discharge are numbered by a slice number. Profiles that are functions of time, are given a slice number

of -1. The database also stores the name of the person or group responsible for each profile.

Many tokamak experiments consists of perturbations made to a plasma that is considered to have reached steady-state. Perturbations such as pellet injections, additional heating methods, etc., are characterized by another set of scalar parameters and their values, called perturbations.

Most of the database entries allow textual comments to be included. These comments have been organized as footnotes and are referenced by a footnote or comment number. These comment numbers, which are returned by several MDI commands, can then be used to retrieve the comments. Note that a comment number of 0 implies "no comment".

Many of the discharges in the database have appeared in published papers or reports. These references are stored in the database and each reference has been given a reference number. The references related to a particular machine can be found as can the discharges associated with a particular reference.

The machines or experimental devices are also described by sets of parameters and values which do not change during a shot and change only when the machine configuration changes. These parameters, which may describe the positions and types of limiters, for example, have two-part names. The parameters are grouped into sets called machine parameter sets. Within each set the parameters are called machine parameters. The database contains a dictionary of most of the words and special terms, which makes the database largely self-defining. Descriptions of coordinate systems and fitting functions are also contained in the dictionary.

Within the database, special variables or column names are used that correspond to the terms used in MDI; for example, a profile is a **physparam**, a characteristic parameter is a **charparam**, a perturbation is a **pertparam**, etc. It is not necessary to know these special names to use MDI; however, it is necessary to use these names in SQL queries.

B. The commands

The MDI commands are organized here according to the type of *Mathematica* object returned and are also approximately in the order in which a new user might wish to explore the database. An alphabetical list of commands can be found in section VI. The MDI commands are highlighted with brief descriptions of their function. The command parameters are usually strings and are set off in double quotation marks. The shot numbers can be given either as strings or integers. The reference numbers and comment numbers should be given as integers. Some parameters are optional and are denoted as such by

being followed by a colon and the default value of the argument. The colon should not be entered. Commands that end in "table" are designed to be used with the `TableForm` output format.

Many commands return a single *Mathematica* list. These lists can be combined with the *Mathematica* list-manipulation commands and other MDI commands to easily generate more complicated queries. Combining these commands with list operations is demonstrated in section III.

A list of all the machines represented in the database can be found with the `MdiMachines` command.

`MdiMachines[]` returns a list of machines in the database.

```
In[2]:= MdiMachines[]

Out[2]= {ASDEX, JET, JT-60, PDX, PLT, T-10, TEXT, TFTR}
```

The published references associated with a particular machine can be found with the `MdiReferences` command.

`MdiReferences["machine"]` returns a list of reference numbers and the references associated with a particular machine.

Several commands return a list of shot numbers that can then be used as input for other MDI commands.

`MdiDischarges["machine"]` returns a list of shot numbers for given machine.
`MdiDischargesFull["machine"]` returns a table of shot numbers for given machine including the characterization data for each discharge.
`MdiDischargesByType["machine", "keyword"]` returns a list of all shot numbers for a machine that are described by the same keyword.
`MdiDischargesByReference["reference number"]` returns a list of machine names and shot numbers associated with a particular reference.

For example, to find all ohmic shot numbers for TEXT the command `MdiDischargesByType["TEXT", "ohmic"]` could be used.

```
In[6]:= MdiDischarges["TEXT"]
Out[6]= {87167, 87590, 87682, 102360, 102479, 107134, 107288, 107411,
110310, 114101, 114519, 121083, 121234}

In[7]:= MdiDischargesByType["TEXT", "Ohmic"]
Out[7]= {87167, 87590, 87682, 102360, 102479, 107134, 107288, 107411,
110310, 114101, 114519, 121083, 121234}
```

The keywords for a discharge can be found with the `MdiKeywords` command.

`MdiKeywords["machine", shot number]` returns a list of keywords describing the discharge. `MdiKeywords["machine"]` returns a list of all unique keywords for the machine.

```
In[8]:= MdiKeywords["JT-60"]
Out[8]= {ICRF, NeutralBeam, Ohmic, Pellet}
```

The keywords are used to broadly classify discharges in types such as diverted, or limited etc. Discharges are also typically characterized by certain parameters.

`MdiDischargeConditions["machine", shot number]` returns the nominal Bt, Ip, NeBar, VLoop, Gas, Zeff, number of time slices for a given discharge, and the comment number in the indicated order.

For example,

```
In[9]:= MdiDischargeConditions["TEXT", 121234]
Out[9]= {2, 200000, 3.0000E+19, 1.518, 1, 1.636, 1, 0}
```

which denotes a 2T magnetic field, 200kA plasma current, etc. discharge. Note that the data in the database is generally in MKS units. For a tabular result which includes the heading see the `MdiDischargeConditionsTable` command.

The machines and discharges are described by additional parameters as well. For example there are parameters which describe the machine configuration, i.e., limiter position and composition.

`MdiMachineParameters["machine"]` returns a list of the names of all the machine parameter sets for the machine.

`MdiMachineParameters["machine", shot number]` returns a list of the names of machine parameter sets and machine parameters and their values for this shot.

`MdiMachineValue["machine", shot number, "machine parameter set", "machine parameter"]` returns the value of a particular machine parameter.

For example, a partial listing of the result of

```
TableForm[MdiMachineParameters["TEXT", 121234]],

gettered   N            0       0
lastvent   20-feb-1988  0       0
limiter    TiC          0       0
limiter    phi          294.5   0
limiter    rmajor       1       0
limiter    rminor       .26     0,
```

shows that this discharge was not gettered, when the machine was last vented, and the position, composition, and size of the limiter. The last column displays the comment number–here indicating no comments were entered.

Discharges are also described by their major and minor radius, the particle and energy confinement times, etc. These parameters are called characteristics and can be accessed with several MDI commands.

`MdiCharacteristics["machine"]` returns a list of all unique characteristic names for the machine.

`MdiCharacteristics["machine", shot number]` returns a list of characteristicc names for the discharge.

`MdiCharacteristicsFull["machine", shot number]` returns a list of characteristic names for the discharge and their values.

`MdiCharacteristicValue["machine", shotnumber, "characteristic"]` returns the value of characteristic name for this discharge.

`MdiCharacteristicsList["machine", shot number, {characteristic1, characteristic2...}]` returns a table of characteristics and their values.

For example,

```
In[10]:= MdiCharacteristics["TEXT", 121234]
Out[10]= {rmajor, rminor, taue, tauee, taup}
```

Discharges are also characterized by the perturbations applied during the discharge.

`MdiPerturbations["machine", shot number]` returns a list of perturbation parameters and their values.

For example, `TableForm[MdiPerturbations["TEXT", 121234]]` returns:

```
DNB    DNBCur     4.7      0
DNB    DNBPulse   .08      0
DNB    DNBTime    .31      0
DNB    DNBVolts   44000    0,
```

which describes the diagnostic neutral beam for this discharge. The last column is the comment number.

The dictionary can be used to lookup any term found in the database.

`MdiDefinition["word"]` returns the definition of word as a string.

For example, `MdiDefinition["TeTS"]` returns:

```
TeTS -  Electron temperature (keV) measured by ThomsScat.
```

The dictionary can be searched for words which match certain patterns.

`MdiDictionary["pattern"]` returns list of words that match the pattern and have definitions in the dictionary.

The wild card character is the "%" symbol. For example to find all words beginning with rx:

```
In[12]:= MdiDictionary["rx%"]
Out[12]= {rxA, rxA, rxI, rxI, rxI, rxS, rxS}
```

Several of the commands return comment numbers. `MdiComment` is used to retrieve the comment.

`MdiComment[`*`comment number`*`]` returns the comment referred to by comment number as a string.

The commands in the next group return formatted tables or reports. They are most often used simply for their results rather than as input to other commands. Note that the database management system, DBMS, generally returns a table complete with column headings. These headings are usually removed by the MDI commands, but the headings from the last query or command can be recovered with the `MdiHeadings` command.

`MdiDischargesTable["`*`machine`*`"]` returns a table of shot numbers with characterization data including column headings. `MdiDischargeConditionsTable["`*`machine`*`", `*`shot number`*`]` returns a table of discharge characteristics with headings for a particular shot. `MdiPerturbationsTable["`*`machine`*`", `*`shot number`*`]` returns a formatted table with headings of the perturbation information.

The primary purpose of the MFEDB is to store profile data, i.e., radial profiles of important tokamak parameters such as the electron temperature. Commands in the next group are used to retrieve and plot profile data. Commands that require a machine name and shot number usually also need a slice number. If a discharge has radial profiles at more than one time, then these profiles are indexed by slice number. Each of these will default to slice = 1, so that the last argument can be omitted if only the first slice is required. Note that profiles that are functions of time have a slice number of -1, so that these functions must have the slice number specified.

Each profile has been given a unique name. The `MdiProfiles` command returns the names of all profiles stored for this discharge.

`MdiProfiles["`*`machine`*`", `*`shot number`*`]` returns a list of profile names for this discharge.

The `MdiProfileData` command returns the name of the independent variable which specifies a particular coordinate system. A description of the coordinate system can be found in the dictionary. The name of the fixed variable and its value is returned. If the profile is stored as a functional fit, the name of the fitting function is given. A comment number is also returned.

`MdiProfileData["machine", shot number, "profile", slice:1]` returns information about the profile.

Information about the person or group responsible for a given profile can be found with `MdiSource` command.

`MdiSource["machine", shot number, "profile"]` returns information about the person or group responsible for this measurement.

The main command for retrieving profile data is the `MdiProfile` command. The commands `MdiProfile` and `MdiProfileXY` return matrices of numbers rather than a list of strings. These matrices can be used directly in *Mathematica* numerical operations and plotting functions.

`MdiProfile["machine", shot number, "profile", slice:1]` returns a list of x, dx, y, dy four-tuples.

```
In[13]:= MdiProfile["TEXT", 121234, "TeTS"]
Out[13]= {{-0.207, 0.01, 0.135, 0.03}, {-0.155, 0.01, 0.294, 0.04},
>    {-0.118, 0.01, 0.453, 0.017}, {-0.066, 0.01, 0.648, 0.026},
>    {-0.002, 0.01, 0.72, 0.045}, {0.06, 0.01, 0.618, 0.024},
>    {0.113, 0.01, 0.426, 0.024}, {0.148, 0.01, 0.307, 0.082},
>    {0.198, 0.01, 0.128, 0.03}}
```

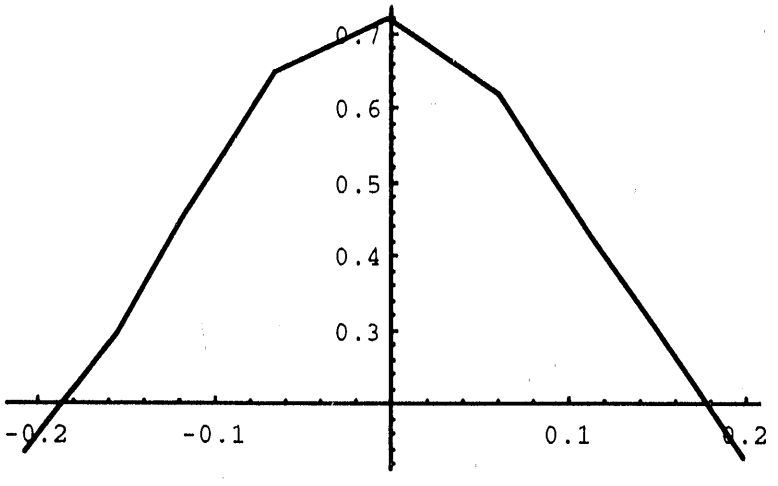
In the cases in which error values have not been provided or only the x-y pairs are required, then the `MdiProfileXY` command can be used.

`MdiProfileXY["machine", shot number, "profile", slice:1]` returns the a list of x-y pairs.

The list can be used directly in the ListPlot command. For example

```
ListPlot[ MdiProfileXY["TEXT", 121234, "TeTS"],
          PlotJoined->True, PlotRange->All];
```

generates a plot of the electron temperature:

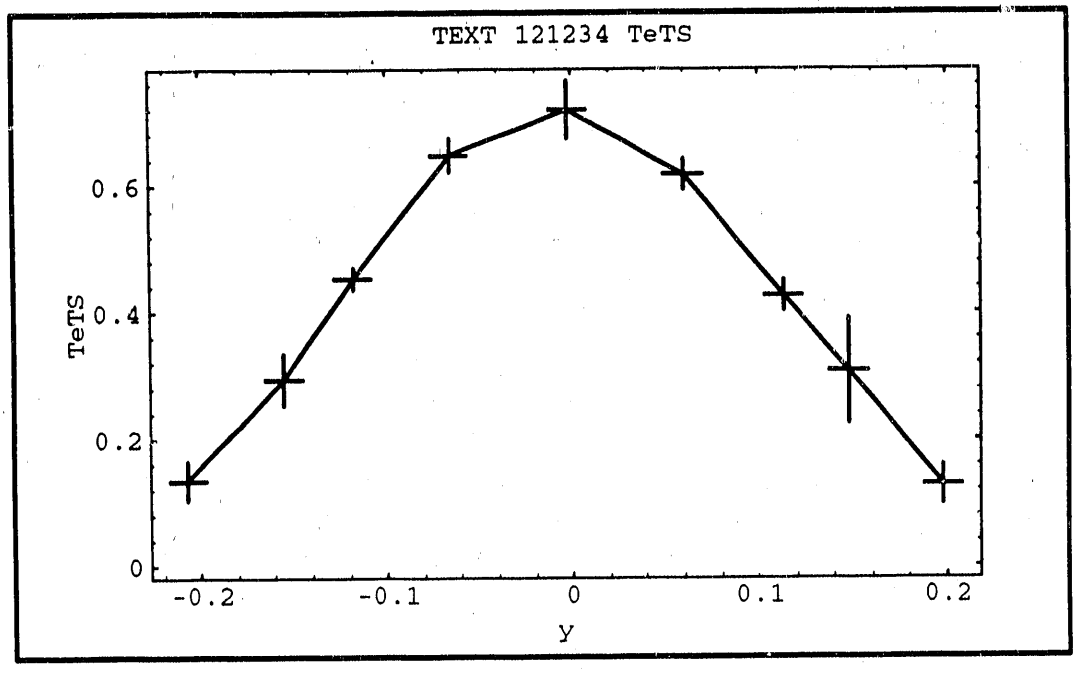


`MdiProfileSize["machine", shot number,"profile", slice:1]` returns the number of points in a profile.

Profiles can be plotted with the axes appropriately labeled with the `MdiProfilePlot` command.

`MdiProfilePlot["machine", shot number, "profile", slice:1]` returns a graphic object including error bars if available.

```
In[16]:= Show[MdiProfilePlot["TEXT", 121234, "TeTS"]]
```



Note that graphics objects rather than plots are returned. `Show` can be used on the result, or else the command can be an argument of `Show`.

Some profiles are described as functions with their function fit parameters stored in the database. These parameters can be retrieved with the `MdiProfileFit` command. The description of the function is stored in the dictionary. The name of the function can be found with the `MdiProfileData` command.

`MdiProfileFit["machine", shot number, "profile", slice:1]` returns the fit parameters.

There are several profiles that are generic to tokamak data and which are necessary for transport analysis. These have their own plot commands so that differences in the profile names and dimensions among the different machines can be ignored. Each of these functions simply calls the `MdiProfilePlot` command with the appropriate profile name for the given machine.

`MdiTePlot["machine", shot number, slice:1]` returns the generic electron temperature profile.

`MdiTiPlot["machine", shot number, slice:1]` returns the generic ion temperature profile.

`MdiNePlot["machine", shot number, slice:1]` returns the generic electron density profile.

`MdiPohPlot["machine", shot number, slice:1]` returns the generic ohmic power profile.

`MdiPauxPlot["machine", shot number, slice:1]` returns the generic auxiliary power profile.

A single page summary of the discharge can be obtained with the command, `MdiSummary`, which collects the nominal parameters and the slice 1 generic profiles.

`MdiSummary["machine", shot number]` returns a graphic array of plots for Te, Ti, ne, Poh, Paux and a table of the nominal discharge parameters.

For generating profile plots within *Mathematica* it is sometimes useful to have the name of the abscissa.

`MdiProfileXLabel["machine", shot number, "profile"]` returns the name of independent variable for the profile.

The *Mathematica* function, `Interpolation`, returns a function that can be evaluated anywhere within the range of the supplied data. It is used to construct the next set of commands.

`MdiProfileIF["machine", shot number, "profile", slice:1]` returns an interpolation function for a profile.

There are special interpolating functions for each of the generic functions.

`MdiTeIF["machine", shot number , slice:1]` returns the interpolating function for generic electron temperature profile.

`MdiTiIF["machine", shot number, slice:1]` returns the interpolating function for generic ion temperature profile.

`MdiNeIF["machine", shot number, slice:1]` returns the interpolating function for generic electron density profile.

`MdiPohIF["machine", shot number, slice:1]` returns the interpolating function for generic ohmic power profile.

`MdiPauxIF["machine", shot number, slice:1]` returns the interpolating function for generic auxiliary power profile.

MDI is not limited to queries that have been preprogrammed either in the MDI package or in the RPC server. Any SQL query can be sent to the database. The query should be a single string and should not end in a semicolon. The RPC transport limits responses to about 200K bytes, so queries should be formulated to keep the query result table less than this amount.

`MdiQuery["Query..."]` submits the query given as a text string to the database and returns the resulting table in *Mathematica* list form.

For example,

```
In[17]:= MdiQuery["select shotnum, ip from discharges where
machine='TEXT' and ip > 290000"]

Out[17]= {{{107411}, {300000}}, {{107288}, {300000}}, {{107134},
{300000}}, {{102479}, {300000}}, {{102360}, {300000}}, {{87682},
{300000}}, {{87590}, {300000}}}.
```

Some retrieval problems which would require many separate queries when done with the *Mathematica* list-processing commands can be done with a single SQL query. In these cases while the MDI commands may be more convenient, the SQL query would be more efficient. The MDI package allows the user to combine both approaches.

If the MDI command returns an error message or a surprising result, it is sometimes useful to examine in more detail the results of the previous command. The following commands provide additional information.

`MdiLastQuery[]` returns the exact form of the last query submitted.

`MdiRows[]` returns the number of rows in the last query result table.

`MdiHeadings[]` returns the DBMS table headings from the last command.

`MdiErrorMessage[]` - If the last query generated an error message from the database engine or from the RPC communication layer, this command can be used to retrieve the full text of the message.

`MdiRawReturn[]` returns the unprocessed result from the *MDIClient* program.

`MdiCommand[]` returns the command line of the last command submitted to the shell.

The output from MDI can be exported to other programs or sent to various output devices in the same way as other *Mathematica* results. There are a few additional commands that simplify some common output operations.

`MdiToSpreadSheet["filename", table name]` writes a data table to an external file in a form suitable for a Macintosh spreadsheet program.

The file will be created in the current directory. The fields are tab delimited, the records are carriage-return delimited, and numbers are converted to `CForm`.

`MdiPSPrint[ -graphic- ]` sends postscript graphics to a particular Apple Laser Writer printer.

The MdiPSPrint command should be modified for each installation.

The `MdiProfileTable` command was constructed to solve a particular database access problem. A listing of the *Mathematica* program for this command can be found in Appendix IV and can be used as a example program for solving data retrieval problems that cannot be done with a single MDI command.

`MdiProfileTable["machine",{shot number list}, {profile list}]` returns a table of profile names for each shot on the list in which each has the same radial points.

III. Examples

The MDI commands provide access to most information in the MFEDB. The true usefulness of MDI is in the combination of MDI commands with the Mathematica extensive list and numerical commands. The examples in this section show some typical uses of MDI.

The MdiSummary command produces a report for a particular discharge. To generate a series of reports for a machine, the following function could be used:

```
machineSummary[m_String, n_Integer:3]:=
  Module[{d, i},
    d = MdiDischarges[m];
    Do[ MdiPSPrint[ MdiSummary[m, d[[i]] ] ], {i, Min[{n, Length[d]}
     ]}];
    Print[StringJoin["Finished summary for", m]]
]
```

If there are 1000 shots for a machine then this command could generate up to 16,000 queries and produce a 1000 page report. The default number is set at 3. The list processing is powerful and should be used with some care.

The interpolation functions can be used to construct other functions, which can then be plotted or operated on by any of the *Mathematica* special functions. For example, the electron thermal velocity function for a discharge can be constructed by:

```
te = MdiTeIF["JT-60",7849];
vte[x_]:=4.19 10^7 Sqrt[ 10^3 te[x] ]
```

The function vte(x) can now be evaluated anywhere within the range of data for that shot and can be used in other *Mathematica* operations.

To plot one nominal characterization parameter against another the following function could be used:

```
ParameterPlot[ m_String, x_Integer, y_Integer]:=
  Module[{t, X, Y},
    t = Transpose[ MdiDischargesFull[m]];
      X = MdiToNumber[ Flatten[ t[[x]] ] ];
      Y = MdiToNumber[ Flatten[ t[[y]] ] ];
      ListPlot[ Transpose[{X,Y}]]
 ]
```

Here x and y are the column numbers in the table returned by `MdiDischargesFull`.

To make a table of all key words for all machines in the database, the following command could be used:

```
Map[{#, MdiKeywords[#]}&, MdiMachines[]]//TableForm.
```

To make a table of the number of slices for each discharge for a machine, define:

```
sliceTable[m_String]:=
      Map[{m, #, Take[MdiDischargeConditions[m,#],{7,7}]}&,
                  MdiDischarges[m]]//TableForm
```

To find the number of shots of a certain type and machine, define:

```
TypedShots[m_String, t_String]:=
  {m, t, Length[MdiDischargesByType[m,t]]}
```

Then to build a table for all machines of that type, use

```
Map[ TypedShots[#, "Ohmic"]&, MdiMachines[]]//TableForm.
```

If one wanted to find all shots in a given reference with a particular type, one could use the command:

```
Intersection[Flatten[Drop[Transpose[MdiDischargesByReference[2]], 1]],
              MdiDischargesByType["TFTR", "ohmic"]].
```

Suppose one wanted to generate a Murakami plot of the TFTR ohmic discharges in the database. The Murakami parameter is n_eR/Bt, and the Murakami plot is the energy confinement time plotted as a function of the Murakami parameter. We first get a list of shots:

```
shots = MdiDischargesByType["TFTR", "Ohmic"];
```

Then we make lists of the major radius, confinement time, and other characteristic parameters:

```
R = MdiToNumber[Map[MdiCharacteristicValue["TFTR",#,"rmajor"]&,shots]];
tau = MdiToNumber[Map[MdiCharacteristicValue["TFTR",#,"TAUE_A_TOI$R"]&,
shots]];
{Bt, Ip, Nebar, Vl, Gas, Zeff, Ts, Cid}=
   Transpose[MdiToNumber[Map[MdiDischargeConditions["TFTR",#]&,shots]]];
```

We next calculate the Murakami parameter for all the shots:

```
Mur = Nebar R /( Bt 10^19);
```

then we make a list of x,y pairs in a form suitable for ListPlot:

```
pts = Transpose[{Flatten[Mur], Flatten[tau]}];
```

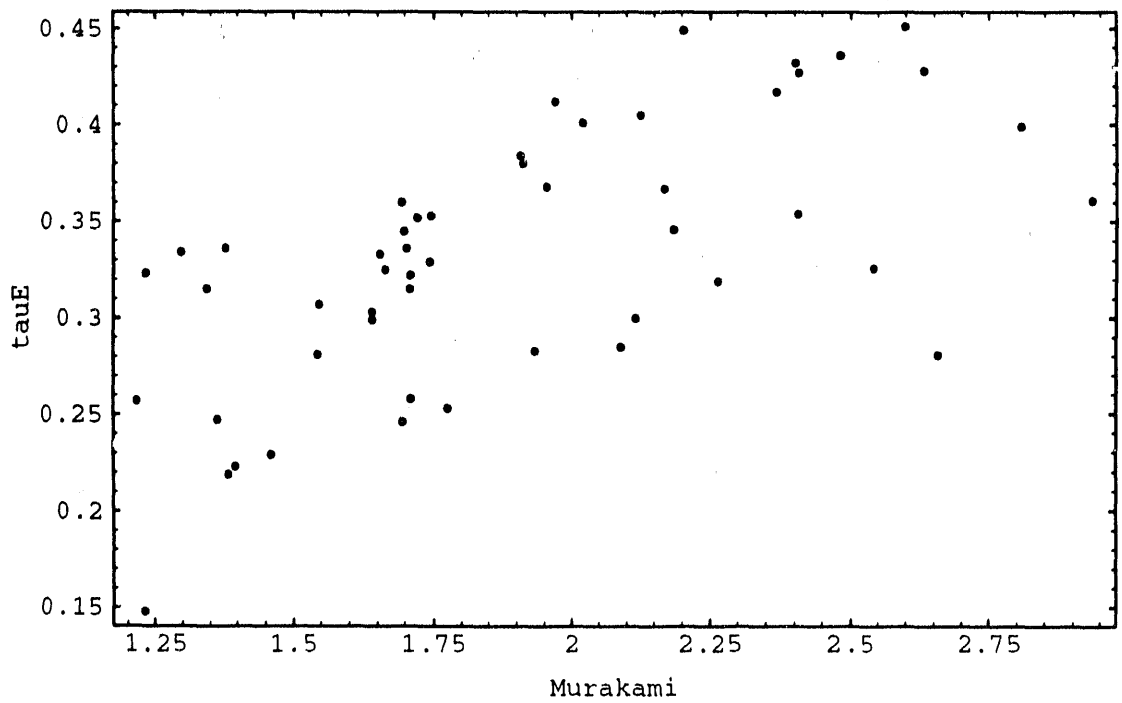
There is one problem because some shots did not have a confinement time available. These shots have EndOfFile as their y value. We simply select those points without this property:

```
npts = Select[pts,#[[2]]!= "EndOfFile"&];
```

We are now ready to make the plot:

```
ListPlot[npts,Frame->True, FrameLabel->{"Murakami","tauE"," "," "}];
```

with the result:



IV. The MDI components

The MDI system is composed of three parts: the Mdi.m package, which adds the database access functions to *Mathematica*, the RPC client/server process pair, which provide network communication between the *Mathematica* kernel and the MFEDB host computer, and the Oracle database manager software, which handles the MFEDB. Figure 1 shows a schematic layout of the MDI components. There are potentially three computers involved and five separate processes. The network connection between a *Mathematica* front-end and the *Mathematica* kernel is controlled by the *Mathematica* software. These two processes can either run on the same computer or be connected through a network or by modem.

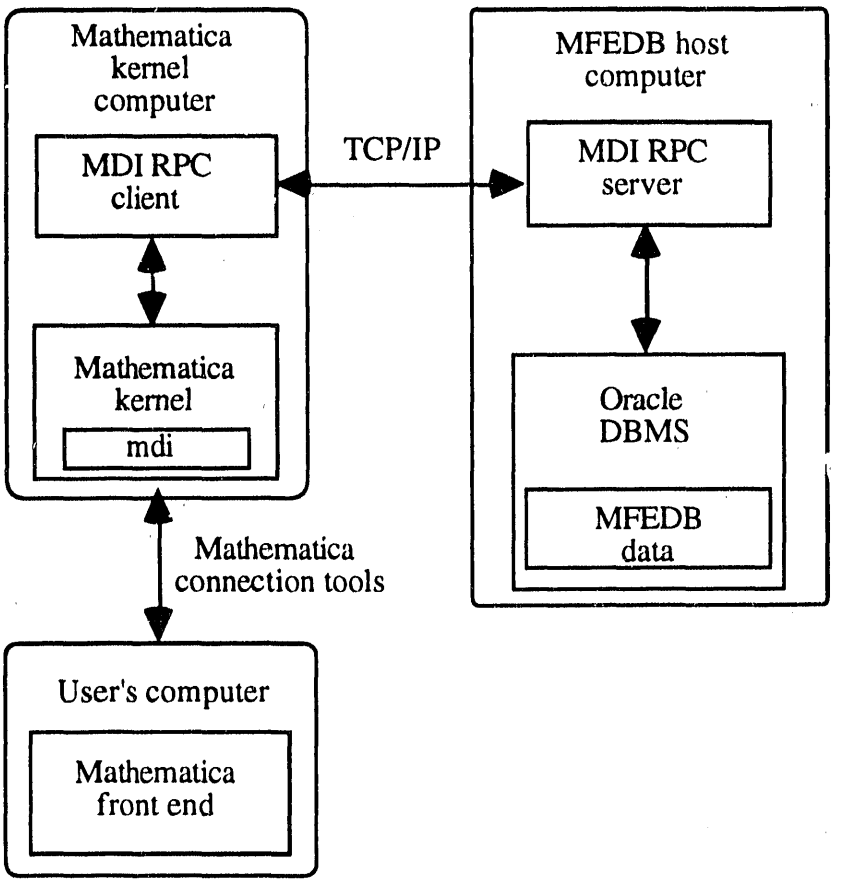


Figure 1. The MDI components.

There are three inter-process communication connections in addition to the *Mathematica* kernel–front-end connection. Two of these connections are between processes running on the same computer, and one connects processes running on different

machines. When data are transmitted between processes there is always the problem of maintaining compatible formats, especially when processes may reside on different computer architectures with different internal data representations. The RPC protocol was designed to solve this problem by including a description of the argument data types in the argument list and defining, through the XDR protocol, standard network data formats. The MDI package largely avoids these problems by exchanging data between processes as text streams.

A second design consideration in multi-process applications is process synchronization. The MDI package is designed as a single execution pipeline, i.e., each process initiates the next process in the sequence and blocks until it either receives a result or times out. Once a process receives a result from the next process in the pipeline it completes its processing and terminates. The chain of process is re-created with each call to an MDI function. An alternate design would require that the MDI package have an open command that would establish the chain of processes to the database manager. MDI commands could then be executed with each of the processes already established, saving the start-up overhead. There would then need to be a close command, which would release each of the processes in the chain. While this design may result in somewhat faster queries, it is significantly more complex and would require more sophisticated programming and error correction at each stage. Simple MDI commands, such as `MdiMachines` execute in about one second, which provides a good estimate of the amount of time required to set up the process chain. More complex queries take longer, but most of this time appears to be due to the database manager and the *Mathematica* processing of the results. For simple applications like MDI the current design appears to be adequate. It can also be viewed as an application prototype with more sophisticated inter-process communication methods added as necessary.

On most computer systems, with the Macintosh being the most notable exception, *Mathematica* can execute an external program from within the *Mathematica* session. There are two types of communication with the external process: structured and unstructured. The structured communication uses the MathLink protocol and allows programs to exchange *Mathematica* expressions of all types. The unstructured communication simply transmits text streams using file read and write operations.

The MDI package on the *Mathematica* kernel computer communicates with the MDI client program using the *Mathematica* function, `Get`. The `Get` function performs unstructured communication with an external program. The argument to the `Get` function contains the complete shell command line to run the MDI client program. This command contains the name of the remote MDI server machine, the integer code of the function to be

performed and a single text string containing the argument to the function. The output of this program, which is written to `stdout`, is piped to *Mathematica* as if *Mathematica* were reading a text file. Because all data is transmitted as text strings, there are no problems with data format or internal data representations. The conversion between numbers and text is done in MDI.

The MDI client program, *MDIClient,* is the network glue that connects the *Mathematica* kernel computer to the MDI RPC service. A listing of *MDIClient* is provided in Appendix II. It performs three functions: checks input parameters, calls the MDI service, and formats the result as *Mathematica* lists. Of the three levels of RPC library routines, MDI uses the lowest level so that the communication protocol can be chosen and the time-out intervals can be set. The TCP rather than the UDP protocol is used to obtain a guaranteed once-delivery message service. The problem with the default UDP protocol is that the database manager may require more time to complete the query than the UDP protocol allows. In this case, the database manager would continue to receive additional requests, each of which would take longer than the allowed time-out interval. The MDI query would then eventually fail even though the query could be performed in the total time-out period allowed. By using the TCP protocol, the database manager is guaranteed to receive the message only once and the time-out period can be controlled. A time-out value of about 90 seconds is typically used. Most queries complete in less than a few seconds so 90 seconds is generally more than adequate. Since MDI is often used in the interactive mode, the time-out period must be long enough to complete the query, but short enough not to waste time if the query or network connection fails.

The *MDIClient* program can be used as an example for the development of other client programs that communicate with the MDI server. At this level there is no requirement for *Mathematica*. Any program that can make RPC calls can use the MDI service.

The *MDIClient* program can be executed from the Unix command line. *Mathematica* is not needed. For example,

```
$MDIClient  hagar.ph.utexas.edu 2 "TeTS"
```

executed on a machine with *MDIClient* installed would return the definition of TeTs enclosed in braces.

V. The Mdi RPC server

The MDI RPC server is an RPC service installed on the MFEDB host computer that responds to a fixed set of commands. All commands have the form of an integer followed by a text string. The integer codes the particular function that the service is to perform. The arguments to that function are contained in the text string. Each function builds and submits a query to the MFEDB. The database manager returns an ASCII coded table. The table is converted into a single text string with embedded carriage-returns as record delimiters. This string is returned as the RPC function return.

The commands are grouped according to the type of their arguments. The constants used to denote each routine are given names here, but these names are only informative. These commands represent a minimum set of SQL queries needed to access most of the data in the database. Note that the terminology used here corresponds to the database internal-column names and not that of MDI, i.e., **physparam** not profile etc.

Functions with text string as the argument:

1. kSQL - The text string contains the full query which is presented to the database manager.

Functions with a single word as an argument:

2. kDefine - The string contains a word to be looked up in the dictionary. A single string containing the definition is returned.

Functions with no arguments:

3. kMachines - A list of machines in the database is returned.

Functions with the machine name as argument:

4. kDischarges.- A list of shot numbers is returned.
5. kDischargesFull - A table of discharges along with the nominal characterization data is returned.
19. kReferences - A list of all references for this machine is returned.
27. kCharparamsAll - A list of all *charparam* names for this machine is returned.
24. kKeywordsAll - A list of all keywords for this machine is returned.
25. kMachineAll - A table of the names of all machine parameters and machine parameters for this machine is returned.

Function with the machine name and keyword as argument:

6. kDischargesTyped - A list of shot numbers is returned.

Functions with the machine name and shot number as arguments:

17. kPhysparams - A list of *physparams* for this discharge is returned.
18. kCharparamFull - A list of *charparams* and values is returned.

7. kCharparam. - A list of *charparams* for this discharge is returned.
8. kNominal - The nominal discharge characterization parameters are returned.
9. kKeywords - A table of keywords and values which describe the discharge is returned.
10. kPerturbations - A table of perturbation information for the discharge is returned.
29. kMachineParam - A table of machine parameter sets, machine parameter names and their values for this shot is returned.

Functions with machine name, shot number, and *charparam* as arguments:

11. kCharparamVal - The value of the particular *charparam* for this discharge is returned.

Functions with machine name, shot number and list of *charparams* as parameters:

12. kCharparamList - A list of values corresponding to the list of *charparams* for the given discharge is returned.

Functions with machine name, shot number, *physparam*, and slice number as arguments:

13. kPhysparam - A four column table with x, dx, y, dy data is returned.
14. kPhysparamXY - A two column table with x,y data is returned.
15. kPhysparamSize - The number of points in the *physparam* is returned.
16. kPhysparamXLabel - The name of the abscissa is returned.
22. kPhysparamData - Information about the *physparam* is returned.
28. kPhysparamFit - Table of fit parameters for this *physparam* is returned.

Functions with machine name, shot number, and *physparam* as arguments:

21. kSource - The person are group responsible for the measurement is returned.

Functions with comment number as an argument:

23. kComment - The comment associated with the comment number is returned.

Functions with reference number as argument:

20. kDischargesByRef - A list of shot numbers and machine names associated with the reference is returned.

Functions with machine name and machine parameter as agruments:

26. kMachineVal - The value of the machine parameter is returned.

A comparsion of the functions provided by this RPC service and the tables in the database as given in Appendix I shows that all the data can be accessed.

The MDI server code communicates with the Oracle database by building a text file containing the necessary "`set`" commands and the query. This file is used as an input file for the SQL*PLUS program, which is executed using the Unix "`system`" command. The output is written into a temporary file that is then processed by the MDI server program and returned to the MDI client program.

A complete listing of the MDI server code is given in Appendix III.

DISCLAIMER

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.

VI. Command Summary

MDI Command Summary

`MdiCharacteristicsFull["machine", shot number]` returns a list of characteristic names for the discharge and their values.

`MdiCharacteristicsList["machine", shot number, {characteristic1, characteristic2...}]` returns a table of characteristics and their values.

`MdiCharacteristics["machine", shot number]` returns a list of characteristic names for the discharge.

`MdiCharacteristics["machine"]` returns a list of all unique characteristic names for the machine.

`MdiCharacteristicValue["machine", shotnumber, "characteristic"]` returns the value of the characteristic for this discharge.

`MdiCommand[]` returns the command line of the last command submitted to the shell.

`MdiComment[comment number]` returns the comment referred to by comment number as a string.

`MdiDefinition["word"]` returns the definition of word as a string.

`MdiDictionary["pattern"]` returns list of words that match the pattern and have definitions in the dictionary.

`MdiDischargesByReference["reference number"]` returns a list of machine names and shot numbers associated with a particular reference.

`MdiDischargesByType["machine", "keyword"]` returns a list of all shot numbers for a machine that are described by the same keyword.

`MdiDischargeConditionsTable["machine", shot number]` returns a table of discharge characteristics with headings for a particular shot.

`MdiDischargeConditions["machine", shot number]` returns the nominal Bt, Ip, NeBar, VLoop, Gas, Zeff, and number of time slices for a given discharge in the indicated order.

`MdiDischargesFull["machine"]` returns a formatted table of shot numbers for given machine including the characterization data for each discharge.

`MdiDischargesTable["machine"]` returns a table of shot numbers with characterization data including column headings.

`MdiDischarges["machine" ]` returns a list of shot numbers for given machine.

`MdiErrorMessage[]` - If the last query generated an error message from the database engine or from the RPC communication layer, this command can be used to retrieve the full text of the message.

`MdiHeadings[]` returns the DBMS table headings from the last command.

`MdiKeywords["machine", shot number]` returns a list of key words describing the discharge.

`MdiKeywords["machine"]` returns a list of all unique key words for the machine.

`MdiLastQuery[]` returns the exact form of the last query submitted.

`MdiMachineParameters["machine", shot number]` returns a list of the names of all machine parameter sets and machine parameters and their values for this shot.

`MdiMachineParameters["machine"]` returns a list of the names of all the names of themachine parameter sets for the machine.

`MdiMachines[]` returns a list of machines in the database.

`MdiMachineValue["machine", shot number, "machine parameter set", "machine parameter"]` returns the value of a particular machine parameter.

`MdiNeIF["machine", shot number, slice:1]` returns the interpolating function for generic electron density profile.

`MdiNePlot["machine", shot number, slice:1]` returns the generic electron density profile.

`MdiPauxIF["machine", shot number, slice:1]` returns the interpolating function for generic auxiliary power profile.

`MdiPauxPlot["machine", shot number, slice:1]` returns the generic auxiliary power profile.

`MdiPerturbationsTable["machine", shot number]` returns a formatted table with headings of the perturbation information.

`MdiPerturbations["machine", shot number]` returns a table of perturbation information.

`MdiPohIF["machine", shot number, slice:1]` returns the interpolating function for generic ohmic power profile.

`MdiPohPlot["machine", shot number, slice:1]` returns the generic ohmic power profile.

`MdiProfileData["machine", shot number, "profile", slice:1]` returns information about the profile.

`MdiProfileFit["machine", shot number, "profile", slice:1]` returns the fit parameters.

`MdiProfileIF["machine", shot number, "profile", slice:1]` returns an interpolation function for a *profile*.

`MdiProfilePlot["machine", shot number, "profile", slice:1]` returns a graphic object including error bars if available.

`MdiProfileSize["machine", shot number, "profile", slice:1]` returns the number of points in a profile.

`MdiProfiles["machine", shot number]` returns a list of profile names for this discharge.

`MdiProfileTable["machine",{shot number list}, {profile list}]` returns a table of profile names for each shot on the list in which each has the same radial points.

`MdiProfileXLabel["machine", shot number, "profile"]` returns the name of independent variable for the profile.

`MdiProfileXY["machine", shot number, "profile", slice:1]` returns the a list of x-y pairs.

`MdiProfile["machine", shot number, "profile", slice:1]` returns a list of x, dx, y, dy four-tuples.

`MdiPSPrint[ -graphic- ]` sends postscript graphics to a particular Apple Laser Writer printer.

`MdiQuery["Query..."]` submits the query given as a text string to the database and returns the resulting table in *Mathematica* list form.

`MdiRawReturn[]` returns the unprocessed result from the *MDIClient* program.

`MdiReferences["machine"]` returns a list of reference numbers and the references associated with a particular machine.

`MdiRows[]` returns the number of rows in the last query result table.

`MdiSource["machine", shot number, "profile"]` returns information about the person or group responsible for this measurement.

`MdiSummary["machine", shot number]` returns a graphic array of plots for Te, Ti, ne, Poh, Paux and a table of the nominal discharge parameters.

`MdiTeIF["machine", shot number , slice:1]` returns the interpolating function for generic electron temperature profile.

`MdiTePlot["machine", shot number, slice:1]` returns the generic electron temperature profile.

`MdiTiIF["machine", shot number, slice:1]` returns the interpolating function for generic ion temperature profile.

`MdiTiPlot["machine", shot number, slice:1]` returns the generic ion temperature profile.

`MdiToSpreadSheet["filename", "data"]` writes data table to an external file in a form suitable for a Macintosh spreadsheet program.

References

[1] S. Wolfram, "Mathematica: a system for doing mathematics", Addison-Wesley, Redwood City, 1991.

[2] W.H. Miner, Jr., J.C. Wiley, D.W. Ross, The University of Texas at Austin, Report FRCR 323, 1988.

[3] SQL Language Reference Manual, Oracle Corp., 1990.

[4] John R. Corbin, "The Art of Distributed Applications", Springer-Verlag, New York, 1990

[5] SQL*Plus User's Guide and Reference, Oracle Corp., 1989.

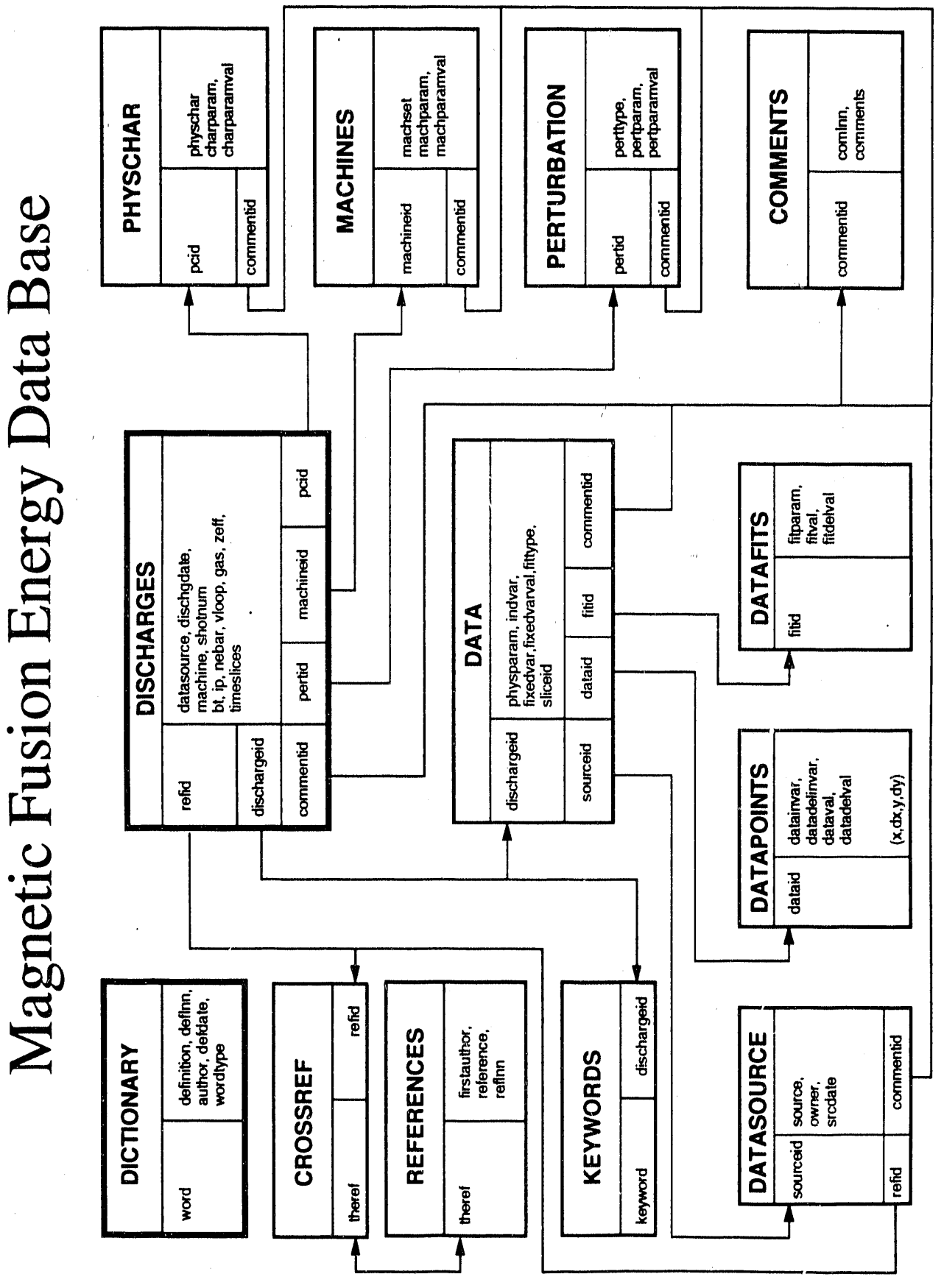
[6] Unix man pages.

[7] Jack J. Dongarra, Eric Grosse, "Distribution of Mathematical Software Via Electronic Mail", netlib, 1985.

[8] W.H. Miner, Jr., R.T. McCann, and J.C. Wiley, Bull. Am. Phys. Soc. **36**, 2436 (1991).

[9] Douglas Comer, "Internetworking with TCP/IP", Prentice-Hall, Englewood Cliffs, 1988.

Appendix I - MFEDB diagram.



Appendix II - The MdiClient code.

Listing 1 - mdiClient.c

```
/*****************************************************************
  -
  - File: mdiClient.c
  -
  - Purpose:  Contacts server on hagar and extrats mfe data.
  -
  - Author: J. C. Wiley, Jan. 1992
  -
  - Notes:
  -        Based on early RPC examples.
  -
  - Change History:
  -
 *****************************************************************/
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <rpc/rpc.h>
#include <netdb.h>
#include <sys/socket.h>
#include <time.h>
#include "mdiServer.h"

 /*****************************************************************
   -
   - mdi_connect - establishes connection with mdi server
   -
 *****************************************************************/
CLIENT *mdi_connect(char *hostname)
  {
    struct hostent    *hostent_p;
    struct sockaddr_in server_addr;
    int                socket;
    char               server_netname[MAXNETNAMELEN+1];
    CLIENT            *clnt_handlep;

    clnt_handlep = NULL;
    if( (hostent_p = gethostbyname(hostname)) != NULL)
      {
        server_addr.sin_family = hostent_p->h_addrtype;
        server_addr.sin_port   = 0;
        socket = RPC_ANYSOCK;
        while( *hostent_p->h_addr_list != NULL) /* check all addresses
*/
          {
            (void)memcpy( (char*)(&server_addr.sin_addr),
                          *hostent_p->h_addr_list,
                           hostent_p->h_length);
            if( (clnt_handlep =
                 clnttcp_create( &server_addr,
                                 MDIRPCPROG, MDIRPCVERS,
                                 &socket, TCPBUFFERSIZE,
                                 TCPBUFFERSIZE))
```

```
                != NULL) break;
            hostent_p->h_addr_list++;
          }
      }
    else
      {
        printf("Could not find host");
      }
    return(clnt_handlep);
  }

 /*****************************************************************
   -
   - toMath - converts string into Mathematica list structure.
   -          This routine assumes that the string is the Oracle
   -          results from the query that is on the first line.
   -          The result is written as a list of lists, each
   -          row of the result table is a list and each token
   -          within the row is an element.
   -
 *****************************************************************/
toMath(unsigned char *data)
  {
    int i,j,k;
    unsigned char *s, *line, *token;
    unsigned char space, newline, quote, squote;

    space   = ' ';
    newline = '\n';
    quote   = '\"';
    squote  = '\'';

    /* replace any double quotes in data with single quotes */
    s = data;
    while(*s != '\0'){ if( *s == quote) *s = squote; s++;}

    /* first line is query, return in first list element */
    s = strtok(data,"\n");
    printf("{\"%s\"},", s);

    /* split remainder of string into lines and then write
       each token as a list element */
    line = (char *)malloc(256);
    token = (char *)malloc(256);
    while( (s = strtok(NULL,"\n")) != NULL)  /* get a line */
      {
         strncpy(line, s, 255);
         i = strlen(line)-1;
         while(i>0 && (line[i] == space)) i--;
         line[i+1] = '\0'; /* strip ending ws */
         /* start */
         printf("{");
         i=0;
         while( line[i] != '\0')
          {
            while(line[i] == space) i++; /* was space */
            j=i;
            while(line[j] != space && line[j] != '\0') j++;
```

```
            j--;
            if(j-i>=0) /* have token */
               for(k=0; k<j-i+1; k++) token[k] = line[i+k];
            token[k]='\0';
            printf("{\"%s\"}", token);
            i=j+1;
            if(line[i] != '\0') printf(",");
           }
         printf("},");
         /* end */
       /*  printf("{\"%s\"},", line);*/
      }
    printf("{}}"); /* takes care of last comma */
    free(line);
    free(token);
  }

typedef struct timeval *TIMEVALP;
/*****************************************************************
  -
  - main
  -
*****************************************************************/
main(int argc, char **argv)
  {
    char *ia;
    int  stat;
    unsigned int len, oper;
    struct send_args sr;
    struct result_args rr;
    SENDPT srpt;
    RESULTPT rrpt;
    CLIENT  *clnt_handlep;
    TIMEVALP timeout;  /* tcp timeout value */
    FILE *logFile;
    time_t t1,t2;
    double ltime;
    char *date;

    date    = (char *)malloc(55);
    srpt = (SENDPT)malloc(sizeof(*srpt));
    rrpt = (RESULTPT)malloc(sizeof(*rrpt));
    timeout = (TIMEVALP)malloc(sizeof(*timeout));
 /*
   -
   - test for valid input parameters.
   -
 */
    printf("{");
    if(argc != 4)
      {
        printf("{usage: mdiClient hostname operation string}");
        printf("}");
        return(1);
      }

    oper = atoi(argv[2]);
    if(oper <=0 || oper > MAXOPER )
```

```
      {
       printf( "{\"operation %d, outside range [1,%d]\"}",
                 oper, MAXOPER);
       printf("}");
       return(1);
      }
/*
  -
  - copy input string into rpc data strucutre
  -
*/
     len = strlen(argv[3])+1;
     ia = (char *)malloc(len);
     if(ia == NULL)
       {
        printf(" memory allocation in client \n");
        return(1);
       }
     strcpy(ia, argv[3]);
     srpt->data = ia;
/*
  -
  - open connection to server.
  -
*/
    clnt_handlep = mdi_connect(argv[1]);
    if(clnt_handlep == NULL)
      {
       fprintf(stdout, "{\"");
       fprintf(stdout,
         "clnttcp_create could not find service ");
       fprintf(stdout, "\"}}");
       return(1);
      }
/*
  -
  - Make call to remote service and log it.
  -
*/
    logFile = fopen("/tmp/mdi/mdiClientLog", "a");
    if(logFile == NULL)
      {
       printf("\"MDIClient: could not open Client logfile\"}}");
       return(1);
      }
    timeout->tv_sec = 95; timeout->tv_usec=0;
    clnt_control(clnt_handlep, CLSET_TIMEOUT, timeout);
    t1 = time(NULL);
    stat = clnt_call(clnt_handlep,
                     (unsigned long)oper,
                      xdr_send, srpt,
                      xdr_result, rrpt,timeout);
    t2 = time(NULL);
    ltime = difftime(t2, t1);
    strftime(date, 25," %m/%d/%y %H:%M:%S ", localtime(&t2));
    fprintf(logFile,"MDIClient: code %ld, status %d, time %f sec %s\n",
                    oper, stat, ltime, date);
    fclose(logFile);
```

```
/*
  -
  - Process error result if necessary.
  -
*/
    if( stat != RPC_SUCCESS )
      {
        fprintf(stdout, "{\"");
        fprintf(stdout, "clnt_call returned with error %d, ",
                            stat);
        switch(stat)
          {
            case RPC_CANTENCODEARGS:
                    printf(" can't encode arguements\n");
                    printf(" len = %d, char ptr %x \n",
                            strlen(srpt->data),srpt->data);
                    break;
            case RPC_CANTDECODERES:
                    printf(" can't decode arguements\n");
                    break;
            case RPC_CANTSEND:
                    printf(" can't send \n");
                    break;
            case RPC_CANTRECV:
                    printf(" can't receive\n");
                    break;
            case RPC_TIMEDOUT:
                    printf(" timed out\n");
                    break;
            default:
                    printf(" clnt_call error code %d\n", stat);
                    break;
          }
        fprintf(stdout, "\"})");
        return(1);
      }

/*  Convert output to Mathematica list */

    toMath(rrpt->data);

/*  Clean up */
    free(ia);
    free(date);
    free(srpt);
    free(rrpt);
    xdr_free(xdr_result, &rr);  /* frees mem allocated by xdr_result */
    return(0);
  }
```

Appendix III - Mdi server listings

Listing 1 - mdiServer.h

```
/*****************************************************************
  -
  - File: mdiServer.h
  -
  - Purpose:  Defines data structures for MDI.
  -
  - Author: J. C. Wiley, Jan. 1992
  -
  - Notes:
  -
  - Change History:
  -
 *****************************************************************/
#define TCPBUFFERSIZE (u_int)8000

/*************** RPC parameters ********************/

#define        MDIRPCPROG        (long)250014
#define        MDIRPCVERS        (long)1
#define        MAXINTALEN        (long)64000
#define        MAXOPER           29
#define        MAXMESSAGE        200000
/*********** XDR structures, etc. ********************/


struct send_args {
                  char *data;
                 };

struct result_args {
                     char *data;
                    };

typedef struct result_args *RESULTPT;
typedef struct send_args *SENDPT;

bool_t xdr_send(XDR *xdrs, SENDPT app)
  {
       return(  xdr_string(xdrs, &app->data, MAXMESSAGE) );
   }

bool_t xdr_result(XDR *xdrs, RESULTPT app)
  {
       return(  xdr_string(xdrs, &app->data, MAXMESSAGE) );
   }
```

Listing 2 - mdiServer.c

```
/*****************************************************************
  -
  - File: mdiServer.c
  -
  - Purpose: These routine perform the RPC requests. This is
  -          a C program because the C++ compiler can not handle
  -          the RPC headers.
  -
  - Author: J. C. Wiley, Jan. 1992
  -
  - Notes:
  -
  - Change History:
  -
 *****************************************************************/
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <rpc/rpc.h>
#include <time.h>
#include "mdiServer.h"

#define assert(ex) {if (!(ex)) \
    {fprintf(logFile,"Assertion failed: line %d of %s: \"%s\"\n", \
    __LINE__, __FILE__, "ex"); \
    fclose(logFile);\
    sleep (30); exit(1);})

 extern char *mdiglue(int code, char *q, FILE *logFile);

/********************** server - effective main *********************/
int server(int Argc, char **Argv)
  {
    SVCXPRT        *TransportHandle;
    void            Dispatch ();

    (TransportHandle = svctcp_create(0,
                                          TCPBUFFERSIZE,
                                          TCPBUFFERSIZE));
/*    pmap_unset (MDIRPCPROG, MDIRPCVERS);*/
    svc_register(TransportHandle, MDIRPCPROG, MDIRPCVERS, Dispatch,
                          0);

    svc_run();
    fprintf (stderr, "exserver: error: svc_run returned\n");
    svc_unregister(MDIRPCPROG, MDIRPCVERS);
    svc_destroy(TransportHandle);
    sleep(30);   /* just in case inetd wants to fork us again */
    return(1);
 }

 /******************* RPC procedures **************/
 void Dispatch(struct svc_req *ServiceRequest,
               SVCXPRT        *Transport)
   {
    RESULTPT resultpt;
```

```
    char xdrbuf[TCPBUFFERSIZE];
    time_t t1,t2;
    double ptime;
    FILE *logFile;
    int code;
    long length;
    char *query;
    char *date;

    date = (char *)malloc(55);
    t1 = time(NULL);
    bzero(xdrbuf, sizeof(xdrbuf) );     /* said to be required... */

    logFile = fopen("/u/peaches/u2/wiley/mdi/bin/mdiServerLog", "a");

    code = ServiceRequest->rq_proc;
    resultpt = (RESULTPT)malloc( sizeof(*resultpt));
    if(code <= 0)
      {
         assert(svc_sendreply(Transport, xdr_void, NULL));
      }
    else if(code <= MAXOPER)
      {

         assert(svc_getargs(Transport, xdr_send, xdrbuf));

         query = ((struct send_args *)xdrbuf)->data;

         resultpt->data = mdiglue(code, query, logFile);

         length = strlen(resultpt->data);

         if(!svc_sendreply(Transport, xdr_result, resultpt))
           {
             t2 = time(NULL);
             ptime = difftime(t2, t1);
             strftime(date, 25," %m/%d/%y %H:%M:%S ",localtime(&t2));
             fprintf(logFile,"MDIServer:svc_sendreply failure: code %d,
len %ld, time %f sec %s\n",
                                  code, length, ptime, date);
             fclose(logFile);
             return;
           }
         assert(svc_freeargs(Transport, xdr_send, xdrbuf));
         free(resultpt->data);
      }
    else
      {
         resultpt->data = (char *)malloc(55);
         sprintf(resultpt->data,
           "MDIServer: unknown function %d called in Dispatch()\n",
code);
         fprintf(logFile,"%s", resultpt->data);
         svc_sendreply(Transport, xdr_result, resultpt);
         free(resultpt->data);
      }
    free(resultpt);
    t2 = time(NULL);
```

```
    ptime = difftime(t2, t1);
    strftime(date, 25," %m/%d/%y %H:%M:%S ",localtime(&t2));
    fprintf(logFile,"MDIServer: code %d, len %ld, time %f sec %s\n",
                                code, length, ptime, date);
    fclose(logFile);
    free(date);
    return;
}
```

Listing 3 - mfedb.h

```
/*****************************************************************************
  -
  - File: mfedb.cxx
  -
  - Purpose: Definition of mfedb class. This class responds to certain
  -        database queries. In general the inputs to the member functions
  -        are text strings and the output is a text string which is the
  -        result of a database query.
  -
  -
  - Author: J. C. Wiley, January 1992.
  -
  - Notes:
  -        The returned string should be deleted by the calling routine
  -
*****************************************************************************/
#ifndef _TMFEDB
#define _TMFEDB

class TMFEDB
  {
      public:
              TMFEDB();
              ~TMFEDB();
              char *fDoQuery(const int code,
                             const char *querystring,
                             FILE *log);
      private:
  };

#endif
```

Listing 4 - mfedb.cxx

```
/*************************************************************************
  -
  - File: mfedb.cxx
  -
  - Purpose: implementation of mfedb class. This class responds to certain
  -        database queries. In general the inputs to the member functions
  -        are text strings and the output is a text string which is the
  -        result of a database query.
  -
  - Author: J. C. Wiley, January 1992.
  -
  - Notes:
*************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "answer.h"
#include "mfedb.h"


/*************************************************************************
  -
  - Constructor
  -
*************************************************************************/
TMFEDB::TMFEDB()
  {

  }

/*************************************************************************
  -
  - Destructor
  -
*************************************************************************/
TMFEDB::~TMFEDB()
  {

  }
/*************************************************************************
  -
  - fDoQuery - queries mfe database
  -
*************************************************************************/
char *TMFEDB::fDoQuery(const int code, const char *query, FILE *log)
  {
      TAnswer *answer;
      char *result;
      char *machine, *discharge, *physparam, *type, *physchar, *word,
*slice;
      char *refid, *commentid, *machineparam, *machineset;
      char **pclist=NULL;
      int n;
      char ws[4] = " ,\t";
      char *copy;      // make copy of query for strtok
```

```
answer = new TAnswer(log);
copy =  new char(strlen(query)+1);
strcpy(copy, query);
switch(code)          // decode arguments
  {
     case kMachines:         // void
            break;
     case kSQL:         // char *
            break;
     case kDefine:         // word
            word = strtok(copy, ws);
            break;
     case kDischarges:         // machine
     case kDischargesFull:
     case kReferences:
     case kKeywordsAll:
     case kMachineAll:
     case kCharparamsAll:
            machine   = strtok(copy, ws);
            break;
     case kDischargesTyped:   // machine, type
            machine = strtok(copy, ws);
            type    = strtok(NULL, ws);
            break;
     case kComment:           // commentid
            commentid = strtok(copy, ws);
            break;
     case kDischargesByRef:   // refid
            refid = strtok(copy, ws);
            break;
     case kCharparam:         // machine, discharge
     case kNominal:
     case kKeywords:
     case kPerturbations:
     case kPhysparams:
     case kCharparamFull:
     case kMachineParam:
            machine   = strtok(copy, ws);
            discharge = strtok(NULL, ws);
            break;
     case kMachineVal:        // machine, discharge, machine set,
machine parameter
            machine   = strtok(copy, ws);
            discharge = strtok(NULL, ws);
            machineset = strtok(NULL, ws);
            machineparam = strtok(NULL, ws);
            break;
     case kCharparamVal:        // machine, discharge, charparam
            machine   = strtok(copy, ws);
            discharge = strtok(NULL, ws);
            physchar  = strtok(NULL, ws);
            break;
     case kCharparamList:         // machine, discharge, list of
charparams
            machine   = strtok(copy, ws);
            discharge = strtok(NULL, ws);
            physchar  = strtok(NULL, ws);
            n = 0;
```

```
            pclist = (char **)malloc( 200 *sizeof(char *));
            while( physchar != NULL && n<200)
              {
                pclist[n++] = physchar;
                physchar = strtok(NULL, ws);
              }
            break;
      case kPhysparam:          // machine, discharge, physparam, slice
      case kPhysparamXY:
      case kPhysparamSize:
      case kPhysparamXLabel:
      case kPhysparamData:
      case kPhysparamFit:
            machine   = strtok(copy, ws);
            discharge = strtok(NULL, ws);
            physparam = strtok(NULL, ws);
            slice     = strtok(NULL, ws);
            break;
     case kSource:            // machine, discharge, physparam
            machine   = strtok(copy, ws);
            discharge = strtok(NULL, ws);
            physparam = strtok(NULL, ws);
            break;
     default:
            answer->fWrite(" Input parse not found for command");
            break;
   }
 switch(code)
   {
     case kSQL:
            answer->fQuery(copy);
            break;
     case kDefine:
            answer->fLookUp(word);
            break;
     case kMachines:
            answer->fMachines();
            break;
     case kDischarges:
            answer->fDischarges(machine);
            break;
     case kDischargesFull:
            answer->fDischargesFull(machine);
            break;
     case kDischargesTyped:
            answer->fDischargesTyped(machine, type);
            break;
     case kCharparam:
            answer->fCharparams(machine, discharge);
            break;
     case kNominal:
            answer->fCharacteristics(machine, discharge);
            break;
     case kKeywords:
            answer->fKeywords(machine, discharge);
            break;
     case kPerturbations:
            answer->fPerturbations(machine, discharge);
```

```
                break;
            case kCharparamVal:
                answer->fCharparamVal(machine, discharge, physchar);
                break;
            case kCharparamList:
                answer->fCharparamList(machine, discharge, n, pclist);
                break;
            case kPhysparam:
                  answer->fPhysparam(machine, discharge, physparam, slice);
                break;
            case kPhysparamXY:
                answer->fPhysparamXY(machine, discharge, physparam, slice);
                break;
            case kPhysparamSize:
                answer->fPhysparamSize(machine, discharge, physparam, slice);
                break;
            case kPhysparamXLabel:
                answer->fPhysparamXLabel(machine, discharge, physparam,
                                                                  slice);
                break;
            case kPhysparams:
                answer->fPhysParams(machine, discharge);
                break;
            case kCharparamFull:
                answer->fCharparamsFull(machine, discharge);
                break;
            case kReferences:
                answer->fReferences(machine);
                break;
            case kDischargesByRef:
                answer->fDischargesByRef(refid);
                break;
            case kSource:
                answer->fSource(machine, discharge, physparam);
                break;
            case kPhysparamData:
                answer->fPhysparamData(machine, discharge, physparam,
slice);
                break;
            case kPhysparamFit:
                answer->fPhysparamFit(machine, discharge, physparam,
slice);
                break;
            case kComment:
                answer->fComment(commentid);
                break;
            case kKeywordsAll:
                answer->fKeywordsAll(machine);
                break;
            case kMachineAll:
                answer->fMachineAll(machine);
                break;
            case kMachineParam:
                answer->fMachineParam(machine, discharge);
                break;
            case kMachineVal:
                answer->fMachineVal(machine, discharge, machineset,
machineparam);
```

```
                break;
        case kCharparamsAll:
                answer->fCharparamsAll(machine);
                break;
        default:
                answer->fWrite(" Function code out of range");
                break;
      }
    result = answer->fGetBuffer();     // copy answer file to buffer
    delete(answer);
    delete(copy);
    if(pclist != NULL) free(pclist);
    return(result);
}
```

Listing 5 - answer.h

```
/*********************************************************************
  -
  - File: answer.h
  -
  - Purpose: does queries on MFE database.
  -
  - Author: J. C. Wiley, January 1992.
  -
  - Notes: Based on netmfe code.
  -
*********************************************************************/
#ifndef _TANSWER
#define _TANSWER
enum {    kSQL=1,
    kDefine=2,
    kMachines=3,
    kDischarges=4,
    kDischargesFull=5,
    kDischargesTyped=6,
    kPhysparams=17,
    kCharparamFull=18,
    kCharparam=7,
    kNominal=8,
    kKeywords=9,
    kPerturbations=10,
    kCharparamVal=11,
    kCharparamList=12,
    kPhysparam=13,
    kPhysparamXY=14,
    kPhysparamSize=15,
    kPhysparamXLabel=16,
    kReferences=19,
    kDischargesByRef=20,
    kSource=21,
    kPhysparamData=22,
    kComment=23,
    kKeywordsAll=24,
    kMachineAll=25,
    kMachineParam=29,
    kMachineVal=26,
    kCharparamsAll=27,
    kPhysparamFit=28
  };
class TAnswer
   {
     public:
        TAnswer(FILE *logFile);
        ~TAnswer(void);
        void fQuery(char *queryString); // 1
        void fLookUp(char *word); // 2
        void fMachines(void); // 3
        void fDischarges(char *machine); // 4
        void fDischargesFull(char *machine); // 5
        void fDischargesTyped(char *machine, char *keyword); // 6
        void fCharparams(char *machine, char *discharge); // 7
        void fCharparamsFull(char *machine, char *discharge); // 18
```

```
        void fCharparamVal(char *machine, char *discharge,
                                                  char *physchar);// 11
        void fCharparamList(char *machine, char *discharge,
                                          int n, char **pclist); // 12
        void fCharparamsAll(char *machine); // kCharparamAll
        void fPhysParams(char *machine, char *discharge); // 17
        void fCharacteristics(char *machine, char *discharge); // 8
        void fKeywords(char *machine, char *discharge); // 9
        void fPerturbations(char *machine, char *discharge); // 10
        void fPhysparam(char *machine, char *shotnum, char *physparam,
                                          char *slice); // 13
        void fPhysparamXY(char *machine, char *shotnum,
                              char *physparam, char *slice); // 14
        void fPhysparamSize(char *machine, char *shotnum,
                              char *physparam, char *slice); // 15
        void fPhysparamXLabel(char *machine, char *shotnum,
                              char *physparam, char *slice); // 16
        void fReferences(char *machine); // kReferences
        void fDischargesByRef(char *refid); //kDischargesByRef
        void fSource(char *machine, char *discharge, char *physparam);
        void fPhysparamData(char *machine, char *discharge,
                        char *physparam, char *slice);
        void fPhysparamFit(char *machine, char *discharge,
                         char *physparam, char *slice);
        void fComment(char *commentid); // kComment
        void fMachineAll(char *machine); //kMachineParam
        void fMachineParam(char *machine, char *discharge);
        void fMachineVal(char *machine, char *discharge, char *set,
                                                    char *param);
        void fKeywordsAll(char *machine); // kKeywordsAll
        void fWrite(char *msg);
        char *fGetBuffer(void); // returns pointer to ans buffer.
    private:
        FILE  *fLogFile;
        long   fFileSize(void);   // returns length of answer message
        void   query(char *theQuery);
        char  *AnswerFile;    /*temp file to build return message */
        char  *fQueryFile;
        char   aQuery[1024];
            };
#endif
```

Listing 6 - answer.cxx

```
/*************************************************************************
  -
  - File: answer.cxx
  -
  - Purpose: Queries mfe database by packaging query into file and making
  -          system call to oracle.
  -
  - Author: J. C. Wiley, Feb. 92.
  -
  - Notes:
  -
  -      Derived from mfedb code, some of which came from netlib code.
*************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "answer.h"


/*************************************************************************
  -
  - constuctor - creates answer file and write TO: line.
  -
*************************************************************************/
TAnswer::TAnswer(FILE *logFile)
  {
    FILE *F;

    fLogFile = logFile;

    AnswerFile = new char[L_tmpnam+1];
    if(AnswerFile == NULL)
      {
        fprintf(fLogFile, "Answer::Answer: out of space\n");
        return;
      }
    tmpnam(AnswerFile);
    F=fopen(AnswerFile,"w");
    if(F == NULL)
      {
        fprintf(fLogFile,"Answer::Answer:could not open temp file: %s\n",
AnswerFile);
        return;
      }
    fclose(F);
  }

/*************************************************************************
  -
  - destructor - closes answer file.
  -
*************************************************************************/
TAnswer::~TAnswer(void)
  {
    remove(AnswerFile);
    delete(AnswerFile);
```

```
  }

/*************************************************************************
  -
  - answer_FileSize - return file size or negative code on error.
  -         -1 - can not open file.
  -         -2 - can not seek on file
  -
*************************************************************************/
long TAnswer::fFileSize(void)
  {
     long   t;
     FILE *fd;

     if( (fd = fopen(AnswerFile, "r")) == NULL )  return(-1);
     if( fseek(fd, 0L, 2) ) /* move to end of file*/
       { fclose(fd);
          return(-2);
       }
     t = ftell(fd); /* get file position */
     fclose(fd);
     return(t);
  }

/*************************************************************************
  -
  - answer_ToBuff - copies answer into buffer
  -
*************************************************************************/
char *TAnswer::fGetBuffer(void)
  {
    FILE *f;
    int inchar;
    long n;
    char *ans;
    char *p;

    n = fFileSize(); // get size of answer file.
    if( n <= 0 ) // answer file error
      {
         ans = new char(255);
         sprintf(ans," fGetBuffer: error opening answer file %ld\n", n);
         return(ans);
      }
    ans = (char *)malloc(n+1);
    p = ans;
    if( ans != NULL)
      {
        f = fopen(AnswerFile, "r");
        while((inchar = fgetc(f)) != EOF) *p++ = (char)inchar;
        *p = 0;
        fclose(f);
      }
    return(ans);
  }

/*************************************************************************
  -
```

```
  - fWrite - writes string to answer file
  -
*************************************************************************/
void TAnswer::fWrite(char *msg)
  {
    FILE  *F;

    F = fopen(AnswerFile, "a");
    fprintf(F, "ANSM: %s\n", msg);
    fclose(F);
  }

/*************************************************************************
  -
  - query - sends query to oracle.
  -
*************************************************************************/
void TAnswer::query(char *theQuery)
  {
    FILE *outQuery, *F;
    char *outQueryName;
    char *queryPtr;
    char buf[256];
    int err;

    F = fopen(AnswerFile, "a");
    /* skip over set commands if necessary */
    queryPtr= strrchr(theQuery,'\n');
    if(queryPtr == NULL) fprintf(F, "ANSQ: %s\n", theQuery);
    else                 fprintf(F, "ANSQ: %s\n", queryPtr+1);
    fclose(F);

    // build modified query file, add page size and tab spec
    //                            check for final semi-colon
    outQueryName = new char[L_tmpnam+1];
    tmpnam(outQueryName);
    outQuery = fopen(outQueryName, "w");
    fprintf(outQuery, "set pagesize 5000\n");
    fprintf(outQuery, "set tab off \n");
    fprintf(outQuery, "set linesize 500\n");
    fprintf(outQuery, "set heading on\n");
    fprintf(outQuery, "set feedback on\n");
    fprintf(outQuery, "set null N/A \n");
    fprintf(outQuery, "%s", theQuery);

    // check if we need to add semiColon
    if( theQuery[strlen(theQuery)] != ';') fprintf(outQuery,";\n");
    fclose(outQuery);

    // build oracle command.
    char s1[]="ORACLE_HOME=/u/hagar/u2/oracle; export ORACLE_HOME; ";
    char s2[]="ORACLE_SID=MFE; export ORACLE_SID; ";
    char s3[]="PATH=$PATH:$ORACLE_HOME/bin; export PATH; ";

    sprintf(buf, "%s %s %s sqlplus -s wiley/wiley < %s  >> %s",
                 s1,s2,s3,            outQueryName,  AnswerFile);
    if( (err=system(buf)) != 0)
      {
```

```
        fprintf(fLogFile," Error in oracle call %d\n",err);
        fflush(fLogFile);
      });
    delete(outQueryName);
  }

/*************************************************************************
  -
  - fQuery - takes input string which contains a SQL query and
  -            makes query file. (1)
  -
*************************************************************************/
void TAnswer::fQuery(char *queryString)
  {
    int p;

    p = strlen(queryString);
    if(queryString[p] == ';') queryString[p]=' ';

    query(queryString);
  }

/*************************************************************************
  -
  -  fLookUp - (2)
  -
*************************************************************************/
void TAnswer::fLookUp(char *word)
  {
    sprintf(aQuery,
    "select word, definition, deflnn from dictionary where\
    Lower(word) = Lower(\'%s\') order by deflnn",
    word);

    query(aQuery);
  }

/*************************************************************************
  -
  -  fMachines - (3)
  -
*************************************************************************/
void TAnswer::fMachines(void)
  {
    sprintf(aQuery,
    "select distinct machine from discharges order by machine");

    query(aQuery);
  }


/*************************************************************************
  -
  -  fDischarges - (4)
  -
*************************************************************************/
void TAnswer::fDischarges(char *machine)
```

```
  {
    sprintf(aQuery,
    "select shotnum from discharges where Lower(machine)=Lower(\'%s\') \
     order by shotnum", machine);

    query(aQuery);
  }

/*************************************************************************
  -
  -  fDischargesFull -  (5)
  -
*************************************************************************/
void TAnswer::fDischargesFull(char *machine)
  {
    sprintf(aQuery,
    "select shotnum, bt, ip, nebar, vloop, gas, timeslices, \
    commentid from discharges where \
    Lower(machine)=Lower(\'%s\') order by shotnum",
    machine);

    query(aQuery);
  }

/*************************************************************************
  -
  -  fDischargesTyped - (6)
  -
*************************************************************************/
void TAnswer::fDischargesTyped(char *machine, char *keyword)
  {
    sprintf(aQuery,
    "select shotnum from discharges d, keywords k where\
    k.dischargeid=d.dischargeid and Lower(d.machine)=Lower(\'%s\')\
    and Lower(k.keyword)=Lower(\'%s\') order by shotnum",
    machine, keyword);

    query(aQuery);
  }

/*************************************************************************
  -
  - fCharparams - return list of physchars. (7)
  -
*************************************************************************/
void TAnswer::fCharparams(char *machine, char *discharge)
  {
    sprintf(aQuery,
    "select charparam from physchar p, discharges d where d.pcid=p.pcid\
    and Lower(d.machine)=Lower(\'%s\') and d.shotnum=%s order by
charparam",
    machine, discharge);

    query(aQuery);
  }

/*************************************************************************
  -
```

```
  - fPhysParams - return list of physparams. (17)
  -
*************************************************************************/
void TAnswer::fPhysParams(char *machine, char *discharge)
  {
    sprintf(aQuery,
    "select distinct da.physparam from data da, discharges d where \
    d.dischargeid = da.dischargeid and \
    Lower(d.machine)=Lower(\'%s\') and d.shotnum=%s order by physparam",
    machine, discharge);

    query(aQuery);
  }

/*************************************************************************
  -
  - fCharparamsFull - return table of charparams and their values. (18)
  -
*************************************************************************/
void TAnswer::fCharparamsFull(char *machine, char *discharge)
  {
    sprintf(aQuery,
    "select p.charparam, p.charparamval, p.commentid from physchar p,
discharges d\
    where d.pcid=p.pcid and Lower(d.machine)=Lower(\'%s\') and d.shotnum=%s
",
    machine, discharge);

    query(aQuery);
  }


/*************************************************************************
  -
  - fCharacteristics - return list of discharge characteristics (8)
  -
*************************************************************************/
void TAnswer::fCharacteristics(char *machine, char *discharge)
  {

    sprintf(aQuery,
    "select  bt, ip, nebar, vloop, gas, zeff, timeslices, commentid from
discharges \
    where Lower(machine)=Lower(\'%s\') and shotnum=%s ",
    machine, discharge);

    query(aQuery);
  }

/*************************************************************************
  -
  - fKeywords - return list of keywords (9)
  -
*************************************************************************/
void TAnswer::fKeywords(char *machine, char *discharge)
  {
    sprintf(aQuery,
    "select keyword from keywords k, discharges d where d.dischargeid=\
```

```
    k.dischargeid and Lower(d.machine)=Lower(\'%s\') and shotnum=%s  \
    order by keyword", machine, discharge);

    query(aQuery);
  }

/*************************************************************************
  -
  - fPerturbations - return list of perturbations (10)
  -
*************************************************************************/
void TAnswer::fPerturbations(char *machine, char *discharge)
  {

    sprintf(aQuery,
    "select  p.perttype, p.pertparam, p.pertparamval, p.commentid from
perturbation p,\
    discharges d where d.pertid=p.pertid and\
    Lower(d.machine)=Lower(\'%s\') and d.shotnum=%s ",
    machine, discharge );

    query(aQuery);
  }

/*************************************************************************
  -
  - fCharparamVal - return value of particular physchar (11)
  -
*************************************************************************/
void TAnswer::fCharparamVal(char *machine, char *discharge, char *physchar)
  {
    sprintf(aQuery,
    "select pc.charparamval from physchar pc, discharges d\
    where d.pcid=pc.pcid and Lower(d.machine)=Lower(\'%s\') and
d.shotnum=%s \
    and Lower(pc.charparam) = Lower(\'%s\')",
    machine, discharge, physchar);

    query(aQuery);
  }

/*************************************************************************
  -
  - fCharparamList - return list of values corresponding to list of
  -                  physchars (12)
  -
*************************************************************************/
void TAnswer::fCharparamList(char *machine, char *discharge, int n,
                             char **pclist)
  {
    int i;
    char tmp[50];

    sprintf(aQuery,
    "select pc.charparam, pc.charparamval from physchar pc, discharges d \
    where d.pcid=pc.pcid and lower(d.machine)=lower(\'%s\') and
d.shotnum=%s and \
    (lower(pc.charparam) = lower(\'%s\') ",
```

```
    machine, discharge, pclist[0]);

    for(i=1; i<n; i++)
      {
        sprintf(tmp," or lower(pc.charparam) = lower(\'%s\')", pclist[i]);
        strcat(aQuery, tmp);
      }
    strcat(aQuery,")");

    query(aQuery);
  }

/*************************************************************************
  -
  - fPhysparam - return physparam data. (13)
  -
*************************************************************************/
void TAnswer::fPhysparam(char *machine, char *shotnum, char *physparam,
                         char *slice)
  {
    sprintf(aQuery,
    "set null 0.0 \n select dp.datainvar, dp.datadelinvar, dp.dataval,\
    dp.datadelval from datapoints dp, data da, discharges d where \
    d.dischargeid=da.dischargeid and da.dataid=dp.dataid and\
    Lower(d.machine)=Lower(\'%s\') and d.shotnum=%s and \
    Lower(da.physparam)=Lower(\'%s\') and da.sliceid = %s",
    machine, shotnum, physparam, slice);

    query(aQuery);
  }

/*************************************************************************
  -
  - fPhysparamXY - return physparam data. (14)
  -
*************************************************************************/
void TAnswer::fPhysparamXY(char *machine, char *shotnum, char *physparam,
                    char *slice)
  {
    sprintf(aQuery,
    "set null 0.0 \n select dp.datainvar, dp.dataval \
    from datapoints dp, data da, discharges d where \
    d.dischargeid=da.dischargeid and da.dataid=dp.dataid and\
    Lower(d.machine)=Lower(\'%s\') and d.shotnum=%s and \
    Lower(da.physparam)=Lower(\'%s\') and da.sliceid = %s",
    machine, shotnum, physparam, slice);

    query(aQuery);
  }

/*************************************************************************
  -
  - fPhysparamFit - return physparam fit.
  -
*************************************************************************/
void TAnswer::fPhysparamFit(char *machine, char *shotnum, char *physparam,
                    char *slice)
  {
```

```
    sprintf(aQuery,
    "set null 0.0 \n select df.fitparam, df.fitval, df.fitdelval \
    from datafits df, data da, discharges d where \
    d.dischargeid=da.dischargeid and da.fitid=df.fitid and\
    Lower(d.machine)=Lower(\'%s\') and d.shotnum=%s and \
    Lower(da.physparam)=Lower(\'%s\') and da.sliceid = %s",
    machine, shotnum, physparam, slice);

    query(aQuery);
  }

/*************************************************************************
  -
  - fPhysparamSize - return size of Physparam in points. (15)
  -
*************************************************************************/
void TAnswer::fPhysparamSize(char *machine, char *shotnum, char *physparam,
                             char *slice)
  {
    sprintf(aQuery,
    "select Count(dp.datainvar) \
    from datapoints dp, data da, discharges d where \
    d.dischargeid=da.dischargeid and da.dataid=dp.dataid and\
    Lower(d.machine)=Lower(\'%s\') and d.shotnum=%s and \
    Lower(da.physparam)=Lower(\'%s\') and da.sliceid = %s",
    machine, shotnum, physparam, slice);

    query(aQuery);
  }

/*************************************************************************
  -
  - fPhysparamXLabel - return physparam abscissa label. (16)
  -
*************************************************************************/
void TAnswer::fPhysparamXLabel(char *machine, char *shotnum, char
*physparam,
                           char *slice)
  {
    sprintf(aQuery,
    "select da.indvar from data da, discharges d \
    where d.dischargeid = da.dischargeid and \
    Lower(d.machine) = Lower(\'%s\') and d.shotnum=%s and \
    Lower(da.physparam) = Lower(\'%s\') and da.sliceid=%s",
    machine, shotnum, physparam, slice);

    query(aQuery);
  }

/*************************************************************************
  -
  - fReferences - return list of references
  -
*************************************************************************/
void TAnswer::fReferences(char *machine)
  {

    sprintf(aQuery,
```

```
    "select unique r.theref, r.firstauthor, r.reference, r.reflnn \
     from discharges d, crossref c, references r where c.refid = d.refid
and \
     c.theref = r.theref and lower(\'%s\') = lower(d.machine) \
     order by r.theref, r.firstauthor, r.reflnn", machine);

     query(aQuery);
  }

/*************************************************************************
  -
  - fDischargesByRef - return list of discharges
  -
*************************************************************************/
void TAnswer::fDischargesByRef(char *theRef)
  {
    sprintf(aQuery,
    "select machine, shotnum from discharges d, crossref c, references r \
     where r.theref = %s and r.theref=c.theref and d.refid=c.refid \
     order by machine, shotnum", theRef);

    query(aQuery);
  }

/*************************************************************************
  -
  - fSource - return list of datasources
  -
*************************************************************************/
void TAnswer::fSource(char *machine, char *discharge, char *physparam)
  {
    sprintf(aQuery,
    "select unique ds.source, ds.owner, ds.srcdate, ds.commentid from
datasource ds, \
     discharges d, data da where ds.sourceid=da.sourceid and \
     lower(da.physparam) = lower(\'%s\') and da.dischargeid=d.dischargeid \
     and d.shotnum = %s and lower(d.machine) = lower(\'%s\')",
     physparam, discharge, machine);

    query(aQuery);
  }

/*************************************************************************
  -
  - fPhysparamData - return information about the physparam
  -
*************************************************************************/
void TAnswer::fPhysparamData(char *machine, char *discharge,
                             char *physparam, char* slice)
  {

    sprintf(aQuery,
    "select da.indvar, da.fixedvar, da.fixedvarval, da.fittype,
da.commentid from data da, \
    discharges d where da.sliceid = %s and lower(da.physparam) = \
    lower(\'%s\') and da.dischargeid=d.dischargeid and d.shotnum = %s \
    and lower(d.machine) = lower(\'%s\')",
    slice, physparam, discharge, machine );
```

```
    query(aQuery);
  }

/***************************************************************************
  -
  - fComment - return a comment
  -
***************************************************************************/
void TAnswer::fComment(char *commentid)
  {

    sprintf(aQuery,
    "select comments from comments where commentid = %s order by comlnn ",
    commentid);

    query(aQuery);
  }

/***************************************************************************
  -
  - fMachineAll - return list of machine parameters
  -
***************************************************************************/
void TAnswer::fMachineAll(char *machine)
  {

    sprintf(aQuery,
    "select unique m.machset from machine m, \
    discharges d where lower(d.machine) = lower(\'%s\') and \
    d.machineid = m.machineid order by m.machset", machine);

    query(aQuery);
  }

/***************************************************************************
  -
  - fMachineParam - return list of machine parameters and their values
  -
***************************************************************************/
void TAnswer::fMachineParam(char *machine, char *discharge)
  {

    sprintf(aQuery,
    "select m.machset, m.machparam, m.machparamval, m.commentid from
machine m, \
    discharges d where lower(d.machine) = lower(\'%s\') and \
    d.machineid = m.machineid and d.shotnum = %s order by m.machset",
    machine, discharge);

    query(aQuery);
  }

/***************************************************************************
  -
  - fMachineVal - return value for particular machine parameter
  -
***************************************************************************/
```

```
void TAnswer::fMachineVal(char *machine, char *discharge,
                          char *machset, char *machparam)
  {

    sprintf(aQuery,
    "select  machparamval from machine m, \
    discharges d where lower(d.machine) = lower(\'%s\') and d.shotnum=%s
and \
    d.machineid = m.machineid and lower(m.machparam) = lower(\'%s\') and \
    lower(m.machset) = lower(\'%s\')",
    machine, discharge, machparam, machset);

    query(aQuery);
  }

/*************************************************************************
  -
  - fKeywordsAll - return all unique keywords for given machine
  -
*************************************************************************/
void TAnswer::fKeywordsAll(char *machine)
  {
    sprintf(aQuery,
    "select unique keyword from keywords k, discharges d where \
    k.dischargeid = d.dischargeid and lower(d.machine)= \
    lower(\'%s\') order by keyword", machine);

    query(aQuery);
  }

/*************************************************************************
  -
  - fCharparamsAll - return all possible charparams for given machine.
  -
*************************************************************************/
void TAnswer::fCharparamsAll(char *machine)
  {

    sprintf(aQuery,
    "select unique charparam from physchar pc, discharges d where \
    pc.pcid = d.pcid and lower(d.machine)=lower(\'%s\') \
    order by charparam", machine);

    query(aQuery);
  }
```

Appendix IV - mdi.m listing

```
(* $Id: MDI.m,v 0.37 04/03/1992  $ *)

(*:Version: Mathematica 2.0 *)

(*:Name: MDI *)

(*:Title: MDI - a Mathematica Database Interface *)

(*:Author:
    James C. Wiley (Fusion Research Center)
    February 1992
*)

(*:Keywords:
    mfedb, database
*)

(*:Requirements:
   Kernal must be executed on RS6000 with the MDIClient
   program in the search path. Uses RPC connections.
*)

(*:Warnings:

*)

(*:Sources:

*)

(*:Summary:
  Routine for extracting data from the mfe database.
  Send Email to netmfe@hagar.ph.utexas.edu
  with the simple message "help" for more information
  about the netmfe database.
  Send comments or questions about mdi to, wiley@calvin.ph.utexas.edu
*)

(*: Notes:
  0.35 - 4/14/92 added rmajor and rminor to MdiSummary
  0.36 - 4/16/92 added ToUpperCase for pp database
  0.37 - 4/16/92 added MdiDictionary command.
  0.38 - 4/21/92 changed NominalCharacteristics to DischargeAttributes
*)

(*-----------------------------------------------------*)

BeginPackage["mdi`"];

Print[
  TableForm[{"MDI - Mathematica Database Interface",
              "J. C. Wiley, W. H. Miner, Jr. and D. W. Ross",
                "Fusion Research Center",
                "University of Texas at Austin",
                     " "
 "preliminary version - please direct questions and comments to:",
```

```
        "wiley@calvin.ph.utexas.edu",
              "or",
       "miner@hagar.ph.utexas.edu"}, TableAlignments->Center]]


MdiVersion::usage = "version number of mdi package"
(***************** Help Package from Document ************)

MdiDictionary::usage = " MdiDictionary[\"regular expression\"] returns list
of
  words in dictionary that are like the regular expression. The wild card
characters
  are % for one are more characters and _ to match a single character."

MdiCharacteristicsList::usage = " MdiCharacteristicsList[\"machine\",shot
number,
  {characteristic1, characteristic2...}] returns a table of characteristics
and
  their values."

MdiCharacteristicValue::usage = " MdiCharacteristicValue[\"machine\",
  shot number, \"characteristic\"] returns the value of characterisitc
  name for this discharge."

MdiCharacteristics::usage = " MdiCharacteristics[\"machine\",  shot number]
  returns a list of characteristic names  for the discharge.\
  MdiCharacteristics[\"machine\"]  returns a list
  of all unique characteristic  names for the machine."

MdiCharacteristicsFull::usage = " MdiCharacteristicsFull[\"machine\",  shot
number]
  returns a list of characteristic names  for the discharge and their
values."

MdiCommand::usage = " MdiCommand[] returns the command line of the last
command
submitted to the shell."

MdiComment::usage = " MdiComment[comment number] return the comment refered
to by
  comment number as a string."

MdiDefinition::usage = " MdiDefinition[\"word\"]  returns the definition
  of word as a string."

MdiDischargesByReference::usage = " MdiDischargesByReference[\"reference
number\"]
returns a list of machine names and shot numbers associated with a
particular\
reference."

MdiDischargesByType::usage = " MdiDischargesByType[\"machine\", \"keyword\"]
returns a list of shot numbers  for a machine all of which are described by
the
same keyword."

MdiDischargesFull::usage = " MdiDischargesFull[\"machine\"]  returns a
formatted
```

```
table of shot number s for given machine including the characterization data
for
each discharge."

MdiDischargesTable::usage = " MdiDischargesTable[\"machine\"] returns a
table of shot numbers with characterization data  including column
headings."

MdiDischarges::usage = " MdiDischarges[\"machine\" ]  returns a list of
shot\
numbers for given machine."

MdiErrorMessage::usage = " MdiErrorMessage[] - If the last query generated
an
error message from the database engine or from the RPC communication layer,
this
command can be used to retrieve the full text of the message."

MdiHeadings::usage = " MdiHeadings[]  returns the DBMS table headings from
the
last command."

MdiKeywords::usage = " MdiKeywords[\"machine\",  shot number]  returns a
list of
key words describing the discharge."

MdiKeywords::usage = " MdiKeywords[\"machine\"]  returns a list of all
unique
key words for the machine."

MdiLastQuery::usage = " MdiLastQuery[]  returns the exact form of the last
query submitted."

MdiMachineParameters::usage = " MdiMachineParameters[\"machine\"]  returns a
list of the names of parameters describing the experimental device that do
not change with shot number and their values."

MdiMachines::usage = " MdiMachines[]   requires no argument and returns a
list
of machines in the database."

MdiMachineValue::usage = " MdiMachineValue[\"machine\", \"machine
parameter\"]
returns the value of a particular machine parameter."

MdiNeIF::usage = " MdiNeIF[\"machine\",  shot number, slice:1]  returns the
interpolating function for generic ne profile. "

MdiNePlot::usage = " MdiNePlot[\"machine\",  shot number, slice:1]   returns
the generic ne profile. "

MdiDischargeConditionsTable::usage = "
MdiDischargeConditionsTable[\"machine\",
shot number]  returns a table  of discharge characteristics with headings
for
a particular shot."

MdiDischargeConditions::usage = " MdiDischargeConditions[\"machine\",
```

```
shot number]  returns the nominal Bt, Ip, NeBar, VLoop, Gas, Zeff, and
number
of time slices for a given discharge in the indicated order."

MdiPauxIF::usage = " MdiPauxIF[\"machine\",  shot number, slice:1] returns
the
interpolating function for generic auxiliary power profile."

MdiPauxPlot::usage = " MdiPauxPlot[\"machine\", shot number, slice:1]
returns
the generic auxiliary power profile."

MdiPerturbationsTable::usage = " MdiPerturbationsTable[\"machine\",  shot
number]
returns a formatted table with headings of the perturbation information."

MdiPerturbations::usage = " MdiPerturbations[\"machine\", shot number]
returns a
table of perturbation information."

MdiPhysParamAtTime::usage = " MdiPhysParamAtTime[ \"machine\", shot number,
proifle,
position]  returns value of profile at given position."

MdiPohIF::usage = " MdiPohIF[\"machine\",  shot number, slice:1] returns the
interpolating function for  generic Ohmic power profile."

MdiPohPlot::usage = " MdiPohPlot[\"machine\", shot number, slice:1] returns
the
generic ohmic power profile."

MdiProfileData::usage = " MdiProfileData[\"machine\",  shot number,
\"profile\"]
returns information about the profile."

MdiProfileIF::usage = " MdiProfileIF[\"machine\", shot number, \"profile\",
slice:1],
returns an interpolation function for a profile."

MdiProfilePlot::usage = " MdiProfilePlot[\"machine\",  shot number,
\"profile\",
slice:1] returns a graphic object  including error bars if available."

MdiProfileSize::usage = " MdiProfileSize[\"machine\",  shot number,
\"profile\",
slice:1]  returns the number of points in a profile."

MdiProfiles::usage = " MdiProfiles[\"machine\",  shot number]  returns a
list of
profile names for this discharge."

MdiProfileTable::usage = " MdiProfileTable[\"machine\", {shot number  list},
{proifle list}]  returns a table of profile names for each shot on the list
in
which each has the same radial points."

MdiProfileXLabel::usage = " MdiProfileXLabel[\"machine\", shot number,
\"profile\"]
```

```
returns the name of independent variable for the profile."

MdiProfileXY::usage = " MdiProfileXY[\"machine\",  shot number, \"profile\",
slice:1]   returns the a list of x-y pairs."

MdiProfileFit::usage = " MdiProfileFit[\"machine\",  shot number,
\"profile\",
slice:1]   returns the a list of profile fit parameters."

MdiProfile::usage = " MdiProfile[\"machine\",  shot number, \"profile\",
slice:1]
returns a list of x, dx, y, dy  four-tuples."

MdiPSPrint::usage = " MdiPSPrint[ -graphic- ]  sends postscript graphics to
a
particular  Apple Laser Writer printer. "

MdiQuery::usage = " MdiQuery[\"Query...\"]  submits the query given as a
text
string to the database and returns the resulting table in Mathematica list
form."

MdiRawReturn::usage = " MdiRawReturn[]  returns the unprocessed result from
the
MDIClient program."

MdiReferences::usage = " MdiReferences[\"machine\"] returns a list of
reference
numbers and the references associated with a particular machine."

MdiRows::usage = " MdiRows[]  returns the number of rows in the last query
result
table."

MdiSource::usage = " MdiSource[\"machine\",  shot number, \"profile\"]
returns
information about the person or group responsible for this measurement."

MdiSummary::usage = " MdiSummary[\"machine\",  shot number]  which returns a
graphic array of plots for Te, Ti, ne, Poh, Paux and a table of the nominal
discharge parameters."

MdiToNumber::usage = "MdiToNumber[\"string\"] - returns number from string."

MdiTeIF::usage = " MdiTeIF[\"machine\",  shot number , slice:1] returns the
interpolating function for generic Te profile."

MdiTePlot::usage = " MdiTePlot[\"machine\",  shot number, slice:1]  returns
the generic Te profile."

MdiTiIF::usage = " MdiTiIF[\"machine\", shot number, slice:1] returns the
interpolating function for generic Ti profile."

MdiTiPlot::usage = " MdiTiPlot[\"machine\",  shot number, slice:1]  returns
the
generic Ti profile."
```

```
MdiToSpreadSheet::usage = " MdiToSpreadSheet[\"filename\", \"data\"] writes
data
table to an external file in a form suitable for a Macintosh spreadsheet
program."

MdiProfileTable::usage =
"      MdiProfileTable[\"machine\", \"discharge list\", \"profile list\"]\n
       Returns a table of profiles for each shot on the list in which
       each has the same radial points"

MdiPhysParamAtPosition::usage =
"      MdiPhysParamAtTime[machine, discharge, profile, position] - \n
       returns value of profile at given position"

Begin["`Private`"]

(* static variables *)

fVersion = 0.38;

fClient = "!/u/peaches/u2/wiley/mdi/bin/MDIClient hagar.ph.utexas.edu ";

fLastQuery = " ";

fMDICommand = " ";

fErrorMessage = " ";

fHeadings = {" "};

fRows = 0;

fRaw = {{}};

blank = " ";

quote ="\"";

     kSQL=1;
     kDefine=2;
     kMachines=3;
     kDischarges=4;
     kDischargesFull=5;
     kDischargesTyped=6;
     kPhysparams=17;
     kCharparamFull=18;
     kCharparam=7;
     kNominal=8;
     kKeywords=9;
     kPerturbations=10;
     kCharparamVal=11;
     kCharparamList=12;
     kPhysparam=13;
     kPhysparamXY=14;
     kPhysparamSize=15;
     kPhysparamXLabel=16;
     kReferences=19;
     kDischargesByRef=20;
```

```
    kSource=21;
    kPhysparamData=22;
    kComment=23;
    kKeywordsAll=24;
    kMachineAll=25;
    kMachineParam=29;
    kMachineVal=26;
    kCharparamsAll=27;
    kPhysparamFit=28;


(* static functions *)


dollarFix[p_String]:= StringReplace[p, ("$"->"\$")]

stringListQ[x_]:= Apply[And, Map[StringQ[#]&,x]]

SetAttributes[MdiToNumber, Listable]

(* the get functions make calls to the MDI rpc service functions
   they are not exported and should not be used outside of mdi*)

getQuery[aQuery_String] :=
        (fMDICommand=StringJoin[fClient, blank, ToString[kSQL], blank,
        quote, dollarFix[aQuery], quote];
        Get[fMDICommand])

getWord[word_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kDefine], blank,
        quote, dollarFix[word], quote];
        Get[fMDICommand])

getMachines[]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kMachines], blank,
        "\"  \" "];
        Get[fMDICommand])

getDischarges[machine_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kDischarges],
        blank, quote, machine, quote];
        Get[fMDICommand])

getDischargesFull[machine_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kDischargesFull],
        blank, quote, machine, quote];
        Get[fMDICommand])

getDischargesTyped[machine_String, type_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kDischargesTyped],
        blank, quote, machine, blank, type, quote];
        Get[fMDICommand])

getCharacteristics[machine_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kCharparamsAll],
        blank, quote, machine, quote];
        Get[fMDICommand])
```

```
getCharacteristics[machine_String, discharge_String]:=
        (fMDICommand=StringJoin[ fClient,blank, ToString[kCharparam], blank,
        quote, machine, blank, discharge, quote];
        Get[fMDICommand])

getCharacteristicsFull[machine_String, discharge_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kCharparamFull],
        blank, quote, machine, blank, discharge, quote];
        Get[fMDICommand])

getCharacteristicVal[machine_String, discharge_String, charparam_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kCharparamVal],
        blank, quote, machine, blank, discharge, blank,
        dollarFix[charparam], quote];
        Get[fMDICommand])

getCharacteristicsList[machine_String, discharge_String,
                        pcStringList_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kCharparamList],
        blank, quote, machine, blank, discharge, blank,
        dollarFix[pcStringList], quote];
        Get[fMDICommand])

getDischargeConditions[machine_String, discharge_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kNominal],blank,
        quote,machine, blank, discharge, quote];
        Get[fMDICommand])

getKeywords[machine_String, discharge_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kKeywords],blank,
        quote, machine, blank, discharge, quote];
        Get[fMDICommand])

getPerturbations[machine_String, discharge_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kPerturbations],
        blank, quote, machine, blank, discharge, quote];
        Get[fMDICommand])

getProfile[machine_String, discharge_String, profile_String,
           slice_Integer:1]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kPhysparam],blank,
        quote, machine, blank, discharge, blank, dollarFix[profile], blank,
        ToString[slice], quote];
        Get[fMDICommand])

getProfileXY[machine_String, discharge_String, profile_String,
             slice_Integer:1]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kPhysparamXY],
        blank, quote, machine, blank, discharge, blank, dollarFix[profile],
        blank, ToString[slice], quote];
        Get[fMDICommand])

getProfileFit[machine_String, discharge_String, profile_String,
              slice_Integer:1]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kPhysparamFit],
        blank, quote, machine, blank, discharge, blank, dollarFix[profile],
        blank, ToString[slice], quote];
        Get[fMDICommand])
```

```
getProfileSize[machine_String, discharge_String, profile_String,
                slice_Integer:1]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kPhysparamSize],
        blank, quote, machine, blank, discharge, blank, dollarFix[profile],
        blank, ToString[slice], quote];
        Get[fMDICommand])

getProfileXLabel[machine_String, discharge_String, profile_String,
                 slice_Integer:1]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kPhysparamXLabel],
        blank, quote, machine, blank, discharge, blank, dollarFix[profile],
        blank, ToString[slice], quote];
        Get[fMDICommand])

getPhysParams[machine_String, discharge_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kPhysparams],
        blank, quote, machine, blank, discharge, quote];
        Get[fMDICommand])

getReferences[machine_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kReferences],
        blank, quote, machine, quote];
        Get[fMDICommand])

getDischargesByReference[refid_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kDischargesByRef],
        blank, quote, refid, quote];
        Get[fMDICommand])

getSource[machine_String, discharge_String, profile_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kSource],blank,
        quote, machine, blank, discharge, blank, dollarFix[profile], quote];
        Get[fMDICommand])

getProfileData[machine_String, discharge_String, profile_String,
                slice_Integer:1]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kPhysparamData],
        blank, quote, machine, blank, discharge, blank, dollarFix[profile],
        blank, ToString[slice], quote];
        Get[fMDICommand])

getComment[commentid_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kComment], blank,
        quote, commentid, quote];
        Get[fMDICommand])

getMachineAll[machine_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kMachineAll],
        blank, quote, machine, quote];
        Get[fMDICommand])

getMachineParameters[machine_String, discharge_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kMachineParam],
        blank, quote, machine, blank, discharge, quote];
        Get[fMDICommand])

getMachineValue[machine_String, discharge_String, machineSet_String,
```

```
                machineParameter_String]:=
        (fMDICommand=StringJoin[ fClient,blank,ToString[kMachineVal],
        blank, quote, machine, blank, discharge, blank,
dollarFix[machineSet],
        blank, dollarFix[machineParameter], quote];
        Get[fMDICommand])

getKeywords[machine_String]:=
        (fMDICommand=StringJoin[ fClient, blank, ToString[kKeywordsAll],
        blank, quote, machine, quote];
        Get[fMDICommand])


(*
   MdiClean takes the raw list returned by MDIClient and tries to determine
   if a query result or error message was returned. If a query is returned
   then the first row should be a string with the query echoed having the
first
   four characters ANSQ. If there is a query returned, then the second to
   last row often contains the message xx rows selected or no rows selected.
   If the query was executed but no rows were returned, then this is
indicated;
   otherwise, the result table is processed and headings are stripped and
   stored. If an error occured in the RPC communication, then an RPC
   error message is returned and the full error message is stored.
*)
MdiClean[x_List]:=
      Module[{y, z, w},
         fRaw = x;                (* store raw return *)
         fErrorMessage = "";
         w = x[[1]];
         If[ StringTake[ w[[1]], 4]=="ANSQ",(* check for error code*)
            (* query ok, any rows? *)
            fLastQuery = w;    (* get query, first row *)
            y = Delete[x, -1];         (* kill last row *)
            If[Length[y]>3,
              z = Flatten[Last[y]];    (* get row with rows selected message
*)
              If[Length[Select[z, #== "selected." &]] == 1,
                   fRows = z[[1]];
                   y = Delete[y, -1],  (* kill rows selected row *)
              ];
              fHeadings = y[[2]];  (* get headings *)
              y = Drop[y, 3];      (* drop first 3 rows, headings *)
              , fRows =0; Return[{"No Rows Returned"}]];
            fRows = Length[y];
            Return[y],

          (* result not query *)
            fErrorMessage= w;
            Return[{"RPC error"}]
         ]
      ]

 (* exported functions *)

 MdiVersion[]:= Return[fVersion]
```

```
MdiToNumber[a_String]:=Read[StringToStream[a], Number]

MdiPSPrint[x_]:=(Display["!psfix  | lpr -Ptechps ",x]; x)

MdiRawReturn[]:= Return[fRaw]

MdiLastQuery[]:= Return[fLastQuery]

MdiHeadings[]:= Return[fHeadings]

MdiRows[]:= Return[fRows]

MdiErrorMessage[]:= Return[fErrorMessage]

MdiCommand[]:= Return[fMDICommand]

(* The following commands map the rpc server into Mathematica with
   only minimal formatting and error correction *)

MdiQuery[aQuery_]:= Module[{result={}},
                  fLastQuery = aQuery;
                  result = MdiClean[ getQuery[aQuery]];
                  Return[result];
                ]

MdiDictionary[re_String]:=
  Flatten[ MdiQuery[
           StringJoin["select word from dictionary where lower(word)
like'",
                        ToLowerCase[re],"'"]
          ]]

MdiDefinition[word_String]:=
  Module[{qr, ln, def},
     qr = MdiClean[getWord[word]];
     def = StringJoin["  ",word," -  "];
     If[ MdiRows[]>0,
         Do[
         ln = Drop[ Drop[ qr[[i]], -1], 1];
         def = StringJoin[def,
          Flatten[
            Transpose[{  Flatten[ln ],
                         Table[" ",{Length[ln ]}]
                       }
           ]]],
           {i, Length[qr]}];,
         def=StringJoin[def,"     No definition found in dictionary"];
     ];
     def
  ]

MdiDefinition2[word_String]:=
  Module[{qr, ln},
     qr = MdiClean[getWord[word]];
     Print[ StringJoin["  ",word," -  "]];
     If[ MdiRows[]>0,
         Do[
         ln = Drop[ Drop[ qr[[i]], -1], 1];
```

```
        Print[StringJoin["      ",
         Flatten[
           Transpose[{  Flatten[ln ],
                        Table[" ",{Length[ln ]}]
                     }
          ]]]],
          {i, Length[qr]}];,
        Print["     No definition found in dictionary"];
    ]
  ]

SetAttributes[MdiDefinition, Listable]

MdiMachines[]:= Flatten[ MdiClean[ getMachines[] ]]

MdiDischarges[machine_String]:=
   Flatten[ MdiClean[ getDischarges[ machine ] ]]

MdiDischargesFull[machine_String]:= MdiClean[ getDischargesFull[ machine ] ]

MdiDischargesByType[machine_String, type_String]:=
   Flatten[ MdiClean[ getDischargesTyped[ machine, type] ]]

MdiDischargesByReference[reference_String]:=
   Partition[Flatten[MdiClean[ getDischargesByReference[reference] ]], 2]

MdiDischargesByReference[reference_Integer]:=
   MdiDischargesByReference[ToString[reference]]

MdiReferences[machine_String]:=
  Module[{qr, row, result={}, ref, refid},
    qr = MdiClean[ getReferences[machine] ];
    If[MdiRows[]==0, Return["None"]];
    Do[
      row = Drop[ qr[[i]], -1]; (* remove reflnn *)
      refid = First[row];
      row = Drop[ row, 1];
      ref = Flatten[
              Transpose[
                (Flatten[row], Table[" ",{Length[row]}])
              ]
            ];
      AppendTo[result, Flatten[{refid, StringJoin[ref]}]],
    {i, Length[qr]}];
    result
  ]


MdiSource[machine_String, discharge_String, profile_String]:=
   MdiClean[ getSource[machine, discharge, profile]]

MdiSource[machine_String, discharge_Integer, profile_String]:=
      MdiSource[machine, ToString[discharge], profile]

MdiProfileData[machine_String, discharge_String, profile_String,
               slice_Integer:1]:=
  Flatten[MdiClean[ getProfileData[machine, discharge, profile, slice]]]
```

```
MdiProfileData[machine_String, discharge_Integer, profile_String,
               slice_Integer:1]:=
   MdiProfileData[machine, ToString[discharge], profile, slice]

MdiComment[n_Integer]:=
Module[{qr, ln, def},
     qr = MdiClean[getComment[ ToString[n]]];
     def = StringJoin["Comment  ",ToString[n]," -  "];
     If[ MdiRows[]>0,
         Do[
         ln = qr;
         def = StringJoin[def,
          Flatten[
            Transpose[{  Flatten[ln ],
                         Table[" ",{Length[ln ]}]
                      }
           ]]],
           {i, Length[qr]}];,
         def=StringJoin[def,"     No definition found in dictionary"];
     ];
     def
  ]

SetAttributes[MdiComment, Listable]

MdiMachineParameters[machine_String]:=
    MdiClean[ getMachineAll[machine] ]

MdiMachineParameters[machine_String, discharge_String]:=
    MdiClean[ getMachineParameters[machine, discharge] ]

MdiMachineParameters[machine_String, discharge_Integer]:=
    MdiMachineParameters[machine, ToString[discharge]]

MdiMachineValue[machine_String, discharge_String,
                machineSet_String, machineParam_String]:=
   Flatten[ MdiClean[ getMachineValue[machine, discharge,
                      machineSet, machineParam] ]]

MdiMachineValue[machine_String, discharge_Integer,
                machineSet_String, machineParam_String]:=
     MdiMachineValue[machine, ToString[discharge],
                machineSet, machineParam]

MdiCharacteristics[machine_String]:=
     Flatten[MdiClean[ getCharacteristics[machine] ]]

MdiCharacteristics[machine_String, discharge_String]:=
     Flatten[ MdiClean[ getCharacteristics[ machine, discharge] ]]

MdiCharacteristics[machine_String, discharge_Integer]:=
     MdiCharacteristics[ machine, ToString[discharge]]

MdiCharacteristicValue[machine_String, discharge_String,
                       charparam_String]:=
     Flatten[ MdiClean[ getCharacteristicVal[machine, discharge,
                                             charparam] ]]
```

```
MdiCharacteristicValue[machine_String, discharge_Integer,
                        charparam_String]:=
     MdiCharacteristicValue[machine, ToString[discharge], charparam]


MdiCharacteristicsFull[machine_String, discharge_String]:=
     MdiClean[getCharacteristicsFull[machine, discharge]]

MdiCharacteristicsFull[machine_String, discharge_Integer]:=
     MdiCharacteristicsFull[ machine, ToString[discharge]]

MdiCharacteristicsList[machine_String, discharge_String,
cpList_?stringListQ]:=
  Module[{pSL, c},
       c=Table[" ",{Length[cpList]}];
       pSL = StringJoin[Flatten[Transpose[{cpList, c}]]];
       MdiClean[ getCharacteristicsList[machine, discharge, pSL]]
    ]

MdiCharacteristicsList[machine_String, discharge_Integer,
cpList_?stringListQ]:=
     MdiCharacteristicsList[machine, ToString[discharge], cpList]

MdiProfiles[machine_String, discharge_String]:=
       Flatten[ MdiClean[ getPhysParams[machine, discharge] ]];

MdiProfiles[machine_String, discharge_Integer]:=
                       MdiProfiles[machine, ToString[discharge]]

MdiDischargeConditions[machine_String, discharge_String]:=
       Flatten[ MdiClean[ getDischargeConditions[machine, discharge] ]]

MdiDischargeConditions[machine_String, discharge_Integer]:=
       MdiDischargeConditions[machine, ToString[discharge]]

MdiDischargeConditionsTable[machine_String, discharge_String] :=
  Module[{h, d},
          d = Flatten[MdiDischargeConditions[machine, discharge]];
          h = Flatten[MdiHeadings[]];
          Transpose[{h,d}]
  ]

MdiDischargeConditionsTable[machine_String, discharge_Integer] :=
    MdiDischargeConditionsTable[machine, ToString[discharge]]

MdiKeywords[machine_String, discharge_String]:=
       Flatten[ MdiClean[ getKeywords[machine, discharge] ]]

MdiKeywords[machine_String, discharge_Integer]:=
       MdiKeywords[machine, ToString[discharge] ]

MdiKeywords[machine_String]:=
    Flatten[ MdiClean[ getKeywords[machine] ]]


MdiPerturbations[machine_String, discharge_String]:=
       MdiClean[ getPerturbations[machine, discharge] ]
```

```
MdiPerturbations[machine_String, discharge_Integer]:=
       MdiPerturbations[machine, ToString[discharge] ]

MdiProfile[machine_String, discharge_String, profile_String,
           slice_Integer:1]:=
  Module[{t},
    t = MdiClean[getProfile[machine, discharge,
                            profile, slice ]];
    If[ MdiRows[]>1,
      Partition[MdiToNumber[Flatten[t]],4],
      Return["no data"]
    ]
  ]

MdiProfile[machine_String, discharge_Integer, profile_String,
           slice_Integer:1]:=
       MdiProfile[ machine, ToString[discharge], profile, slice]

MdiProfileXY[machine_String, discharge_String, profile_String,
             slice_Integer:1]:=
  Module[{t},
    t = MdiClean[getProfileXY[machine, discharge,
                            profile, slice ]];
    If[ MdiRows[]>1,
      Partition[MdiToNumber[Flatten[t]],2],
      Return["no data"]
    ]
  ]

MdiProfileXY[machine_String, discharge_Integer, profile_String,
             slice_Integer:1]:=
       MdiProfileXY[ machine, ToString[discharge], profile, slice]


MdiProfileSize[machine_String, discharge_String, profile_String,
               slice_Integer:1]:=
       Flatten[ MdiClean[ getProfileSize[machine, discharge,
                            profile, slice] ]];

MdiProfileSize[machine_String, discharge_Integer, profile_String,
    slice_Integer:1]:=
    MdiProfileSize[machine, ToString[discharge], profile, slice]

MdiProfileFit[machine_String, discharge_String, profile_String,
       slice_Integer:1]:=
       Flatten[ MdiClean[ getProfileFit[machine, discharge,
                            profile, slice] ]];

MdiProfileFit[machine_String, discharge_Integer, profile_String,
              slice_Integer:1]:=
    MdiProfileFit[machine, ToString[discharge], profile, slice]


MdiProfileXLabel[machine_String, discharge_String, profile_String,
                 slice_Integer:1]:=
       Flatten[ MdiClean[ getProfileXLabel[machine, discharge,
                            profile, slice] ]];
```

```
MdiProfileXLabel[machine_String, discharge_Integer, profile_String,
                 slice_Integer:1]:=
      MdiProfileXLabel[ machine, ToString[discharge], profile, slice]


(* The following commands use the basic Mdi commands to construct new
   functions *)

MdiDischargesTable[machine_String]:=
                 Module[{t, h},
                   t = MdiDischargesFull[ machine ];
                   h = MdiHeadings[];
                    Insert[t,h,1]
                 ]

MdiPerturbationsTable[machine_String, discharge_String]:=
    Module[{t, h},
           t = MdiPerturbations[ machine, discharge ];
           h = MdiHeadings[];
           Insert[t,h,1]
    ]

MdiPerturbationsTable[machine_String, discharge_Integer]:=
       MdiPerturbationsTable[machine, ToString[discharge] ]


MdiProfileIF[machine_String, discharge_String, profile_String,
  slice_Integer:1]:=
  Module[{t, z, s},
    t = MdiClean[getProfile[machine, discharge,
                            profile, slice ]];
    If[ MdiRows[]>1,
      z = Partition[MdiToNumber[Flatten[t]],4];
      s = Transpose[z];
      Interpolation[Transpose[{s[[1]], s[[3]]}]],
      Return["no data"]
    ]
  ]

MdiProfileIF[machine_String, discharge_Integer, profile_String,
             slice_Integer:1]:=
   MdiProfileIF[ machine, ToString[discharge], profile, slice]


g[{x_, dx_, y_, dy_}]:= ({{x , y+dy},{x, y-dy}},{{x+dx , y},{x-dx, y}})

boundary[]:= Line[ Map[ Scaled,{{0.01, 0.01},{0.99, 0.01},{0.99, 0.99},
                                {0.01, 0.99},{0.01, 0.01}}]]

MdiProfilePlot[machine_String, discharge_String, profile_String,
               slice_Integer:1]:=
     Module[{t, z, s, xLabel, scal, fac, yLabel},
       z = MdiProfile[machine, discharge, profile, slice];
       If[ MdiRows[]>1,
         (*z = Partition[MdiToNumber[Flatten[t]],4];*)
         s = Transpose[z];
         scal = Floor[N[Log[10, Max[Abs[s[[3]]]]]]];
         If[ Abs[scal]> 2,
```

```
            fac = N[ 10^scal];
            s[[3]] = s[[3]]/fac;
            s[[4]] = s[[4]]/fac;
            z = Transpose[s];
            yLabel = StringJoin[profile," (10^",ToString[scal],")"],
            yLabel = profile
        ];
        xLabel = First[MdiProfileXLabel[machine, discharge, profile,
                                        slice]];
        Graphics[
         {Rectangle[{0.02, 0.02}, {0.97, 0.97},
              Graphics[
                {
                  Line[ Transpose[{s[[1]],s[[3]]}]],
                  Map[ Line, Flatten[Map[g, z], 1]]
                 },
                Frame->True,
                FrameLabel->{xLabel, yLabel,
                  StringJoin[machine," ",discharge," ",profile], " "}
              ]
           ],
           Rectangle[{0,0}, {1,1}, Graphics[boundary[]]]
         }],
        Return[
             Graphics[{boundary[],
                 Text[StringJoin["no Data for ", profile],
                      Scaled[{0.5, 0.5}],{0, 0}]
             }]
        ];
      ]
    ]

MdiProfilePlot[machine_String, discharge_Integer, profile_String,
               slice_Integer:1]:=
        MdiProfilePlot[machine, ToString[discharge], profile, slice]

(* define database of profiles for each of the generic functions *)

ppTe["ASDEX"]= "TR_TEUSE";
ppTe["JET"]  = "TR_TEUSE";
ppTe["JT-60"]= "TE_JT-60";
ppTe["PDX"]  = "te_FISH";
ppTe["PLT"]  = "te_nbtemp";
ppTe["T-10"] = "TE_T-10";
ppTe["TEXT"] = "TeTS";
ppTe["TFTR"] = "TEMP$RA(1)";
ppTe[x_]     = "none";

ppTi["ASDEX"]="TR_TIBAL";
ppTi["JET"]  ="TR_TI";
ppTi["JT-60"]="TI_JT-60";
ppTi["PDX"]  = "none";
ppTi["PLT"]  = "none";
ppTi["T-10"] ="TI_T-10";
ppTi["TEXT"] ="TiImp";
ppTi["TFTR"] ="TEMP$RA(2)";
ppTi[x_]     = "none";
```

```
ppNe["ASDEX"]="TR_NEUSE";
ppNe["JET"]   ="TR_NEUSE";
ppNe["JT-60"]="NE_JT-60";
ppNe["PDX"]   ="ne_FISH";
ppNe["PLT"]   ="ne_nbtemp";
ppNe["T-10"]  ="NE_T-10";
ppNe["TEXT"]  ="neFIR";
ppNe["TFTR"]  ="DEN$RA(1)";
ppNe[x_]      = "none";

ppPaux["ASDEX"]="Paux";
ppPaux["JET"]   ="Paux";
ppPaux["JT-60"]="PBI_JT-60";
ppPaux["PDX"]   ="Paux";
ppPaux["PLT"]   ="Paux";
ppPaux["T-10"]  ="PET-10";
ppPaux["TEXT"]  ="Paux";
ppPaux["TFTR"]  ="PBI$RA";
ppPaux[x_] = "none";

ppPoh["ASDEX"]="Poh";
ppPoh["JET"]   ="Poh";
ppPoh["JT-60"]="PR_JT-60";
ppPoh["PDX"]   ="Poh";
ppPoh["PLT"]   ="Poh";
ppPoh["T-10"]  ="Poh";
ppPoh["TEXT"]  ="Poh";
ppPoh["TFTR"]  ="POH_PROFILE$RA";
ppPoh[x_]      = "none";

MdiTePlot[machine_String, discharge_String, slice_Integer:1]:=
       MdiProfilePlot[machine, discharge, ppTe[ToUpperCase[machine]], slice]

MdiTePlot[machine_String, discharge_Integer, slice_Integer:1]:=
              MdiTePlot[machine, ToString[discharge], slice]

MdiTiPlot[machine_String, discharge_String, slice_Integer:1]:=
       MdiProfilePlot[machine, discharge, ppTi[ToUpperCase[machine]], slice]

MdiTiPlot[machine_String, discharge_Integer, slice_Integer:1]:=
              MdiTiPlot[machine, ToString[discharge], slice]

MdiNePlot[machine_String, discharge_String, slice_Integer:1]:=
        MdiProfilePlot[machine, discharge, ppNe[ToUpperCase[machine]],
slice]

MdiNePlot[machine_String, discharge_Integer, slice_Integer:1]:=
              MdiNePlot[machine, ToString[discharge], slice]

MdiPohPlot[machine_String, discharge_String, slice_Integer:1]:=
        MdiProfilePlot[machine, discharge, ppPoh[ToUpperCase[machine]],
slice]

MdiPohPlot[machine_String, discharge_Integer, slice_Integer:1]:=
     MdiPohPlot[machine, ToString[discharge], slice]

MdiPauxPlot[machine_String, discharge_String, slice_Integer:1]:=
```

```
        MdiProfilePlot[machine, discharge, ppPaux[ToUpperCase[machine]],
slice]

MdiPauxPlot[machine_String, discharge_Integer, slice_Integer:1]:=
     MdiPauxPlot[machine, ToString[discharge], slice]

MdiTeIF[machine_String, discharge_String, slice_Integer:1]:=
       MdiProfileIF[machine, discharge, ppTe[ToUpperCase[machine]], slice]

MdiTeIF[machine_String, discharge_Integer, slice_Integer:1]:=
       MdiTeIF[machine, ToString[discharge], slice]

MdiTiIF[machine_String, discharge_String, slice_Integer:1]:=
       MdiProfileIF[machine, discharge, ppTi[ToUpperCase[machine]], slice]

MdiTiIF[machine_String, discharge_Integer, slice_Integer:1]:=
       MdiTiIF[machine, ToString[discharge], slice]

MdiNeIF[machine_String, discharge_String, slice_Integer:1]:=
       MdiProfileIF[machine, discharge, ppNe[ToUpperCase[machine]], slice]

MdiNeIF[machine_String, discharge_Integer, slice_Integer:1]:=
              MdiNeIF[machine, ToString[discharge], slice]

MdiPohIF[machine_String, discharge_String, slice_Integer:1]:=
        MdiProfileIF[machine, discharge, ppPoh[ToUpperCase[machine]], slice]

MdiPohIF[machine_String, discharge_Integer, slice_Integer:1]:=
     MdiPohIF[machine, ToString[discharge], slice]

MdiPauxIF[machine_String, discharge_String, slice_Integer:1]:=
        MdiProfileIF[machine, discharge, ppPaux[ToUpperCase[machine]],
slice]

MdiPauxIF[machine_String, discharge_Integer, slice_Integer:1]:=
     MdiPauxIF[machine, ToString[discharge], slice]

MdiSummary[machine_String, discharge_String]:=
      GraphicsArray[
         {
           {Graphics[{boundary[],
                   Text[StringJoin[machine," ",discharge],
                        Scaled[{0.5, 0.97}],{0,1}],
                   Text[ Append[Append[Append[
                           Drop[MdiDischargeConditionsTable[machine,
                                                   discharge],-2],
                    {"rmajor", MdiCharacteristicValue[machine,discharge,
                           "rmajor"]}],
                    {"rminor", MdiCharacteristicValue[machine,discharge,
                           "rminor"]}],
                      MdiKeywords[machine,discharge] ]//TableForm,
                         Scaled[{0.05, 0.85}],{-1,1}]
                     }], MdiTePlot[machine, discharge]},
           {MdiNePlot[machine, discharge],  MdiTiPlot[machine, discharge]},
           {MdiPohPlot[machine, discharge], MdiPauxPlot[machine,discharge]}
         }
      ]
MdiSummary[machine_String, discharge_Integer]:=
```

```
      MdiSummary[machine, ToString[discharge]]


MdiPhysParamAtPosition[machine_String, discharge_String, profile_String,
                       place_Real]:=
  Flatten[
    MdiQuery[
      StringJoin[
        "select dp.dataval from data da, datapoints dp, discharges d ",
              "where da.dischargeid = d.dischargeid and ",
              "Lower('",machine, "') = Lower(d.machine) and ",
              "d.shotnum = ", discharge,
              " and da.profile = '", profile,"' and ",
              "dp.datainvar = ", ToString[place],
              " and da.dataid = dp.dataid"
      ]
    ]
  ]

MdiPhysParamAtPosition[machine_String,   discharge_Integer,
                       profile_String, place_Real]:=
         MdiPhysParamAtPosition[machine, ToString[discharge],
                                profile, place]


MdiToSpreadSheet[filename_String, data_List]:=
   Module[{file},
      file = OpenWrite[filename];
      Scan[
        ( Scan[ WriteString[file, CForm[#], "\t"]&, # ];
          WriteString[file,"\n"])&,
        data
      ];
      Close[file];
   ]

MdiCharacteristicsTable[machine_String, shotList_List, pcList_List]:=
  Module[{npc, nshots, rt, qr, key, value, shot, i, j},
    npc = Length[pcList];
    nshots = Length[shotList];
    rt = Table["N/A", {npc+1},{nshots+1}];
    rt[[1,1]] = "shot";
    For[ i =1, i <= npc, i++, rt[[i+1,1]] = pcList[[i]] ];
    Do[
       shot = shotList[[j]];
       rt[[1, j+1]] = shot;  (* put shot as first col *)
       qr = MdiCharacteristics[machine, shot, pcList];
       Do[
          {{key}, {value}} = qr[[i]];
           el = Position[pcList, key][[1,1]];
           If[ IntegerQ[ el ], rt[[ el+1, j+1]] = value,],
           {i, Length[qr]}
       ],
       {j, nshots}
    ];
    Return[rt]
  ]
```

```
MdiProfileTable[machine_String, shotList_List, ppList_List]:=
  Module[(npp = Length[ppList],
          nshots = Length[shotList],
          rt,
          index=0, shot, pp, nr=0, x, y, z, i, j, k, l),
    (* Need to retrieve first case to find number of points
       and their positions *)
    pp = ppList[[1]];
    While[index < nshots,
       index++;
       shot = shotList[[index]];
       z = MdiProfileXY[machine, shot, pp];
       nr = Length[z];
       If[nr >1, Break[], ]
    ];
    If[nr < 2, Return["noEntries Found"],];
    (* Build Empty Table *)
    rt = Table["N/A", (npp nshots +1),(2 + nr)];
    rt[[1, 1]] = "Discharge";
    rt[[1, 2]] = "PhysParam\\R";
    x = Flatten[Take[Transpose[z],1]]; (* get x values *)
    Do[ rt[[1,l+2]]= x[[l]], (l, nr)]; (* x headers *)
    index = 1;
    Do[
       shot = shotList[[i]];
       Do[
         index++;
         pp = ppList[[j]];
         rt[[index, 1]] = shot;
         rt[[index, 2]] = pp;
         z = MdiProfileXY[machine, shot, pp ];
         If[ Length[z] == nr,
            (x, y) = Transpose[z];
            Do[
              If[  rt[[1, k+2]] == x[[k]],
               rt[[index, k+2]] = ToString[FortranForm[y[[k]]]] ],
            (k, nr)]
         ],
       (j, npp)],
    (i, nshots)];
    rt
  ]

SetAttributes[(MdiCharacteristics, MdiComment, MdiDefinition,
               MdiDischargesByReference, MdiDischargesFull,
               MdiDischargesTable, MdiDischarges, MdiKeywords,
               MdiMachineParameters, MdiReferences), Listable]

End[]

Protect[
MdiCharacteristics,                MdiPhysParamAtTime,
MdiCharacteristicsFull,            MdiPohIF,
MdiCharacteristicsList,            MdiPohPlot,
MdiCharacteristicValue,            MdiProfile,
MdiCommand,                        MdiProfileData,
MdiComment,                        MdiProfileFit,
MdiDefinition,                     MdiProfileIF,
```

```
MdiDictionary,
MdiDischarges,
MdiDischargesByReference,
MdiDischargesByType,
MdiDischargesFull,
MdiDischargesTable,
MdiErrorMessage,
MdiHeadings,
MdiKeywords,
MdiLastQuery,
MdiMachineParameters,
MdiMachines,
MdiMachineValue,
MdiNeIF,
MdiNePlot,
MdiDischargeConditions,
MdiDischargeConditionsTable,
MdiPauxIF,
MdiPauxPlot,
MdiPerturbations,
MdiPerturbationsTable,
MdiProfilePlot,
MdiProfiles,
MdiProfileSize,
MdiProfileTable,
MdiProfileXLabel,
MdiProfileXY,
MdiPSPrint,
MdiQuery,
MdiRawReturn,
MdiReferences,
MdiRows,
MdiSource,
MdiSummary,
MdiTeIF,
MdiTePlot,
MdiTiIF,
MdiTiPlot,
MdiToNumber,
MdiToSpreadSheet,
MdiVersion        ]

EndPackage[]
```

END

DATE FILMED

7 / 20 / 92