58/13-9395(2)

PREPARED FOR THE U.S. DEPARTMENT OF ENERGY,
UNDER CONTRACT DE-AC02-76-CHO-3073

PPPL-2871
UC-426,427

PPPL-2871

DATA-SHARING PROTOCOL
A PROTOTYPE IMPLEMENTATION

BY

T. GIBNEY AND D. GREENWOOD

DECEMBER, 1992

PPPL

PRINCETON
PLASMA PHYSICS
LABORATORY

PRINCETON UNIVERSITY, PRINCETON, NEW JERSEY

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

NOTICE

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 produce, 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.

NOTICE

This report has been reproduced from the best available copy.
Available in paper copy and microfiche.

Number of pages in this report: 15

DOE and DOE contractors can obtain copies of this report from:

Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831;
(615) 576-8401.

This report is publicly available from the:

National Technical Information Service
Department of Commerce
5285 Port Royal Road
Springfield, Virginia 22161
(703) 487-4650

Data-Sharing Protocol

A prototype implementation

PPPL--2871

DE93 005729

by
Thomas Gibney, PPPL
David Greenwood, ORNL

1. Introduction

This paper describes a client/server communication protocol which will allow physicists to access data from cooperating remote experiments. Special low-level "client" software within the user's "home" data-access library formulates a request for data from the remote experiment. This request is sent over a network to a server at the remote site. The server has specific knowledge about the location and format of the requested data. The server gets the data and sends it over the network to the requesting client, which reformats the data according to the local library's conventions.

Our prototype is being developed to support remote access to data from ATF, PBX, and microVax data from Tore-Supra. We have attempted to create a flexible design which should accomodate data from other experiments as well.

2. The client-server relationship

It is important to realize that this is *not* a specification for a general client-server system. Rather, it is a definition of a protocol for network communication, geared specifically to the type of data which we encounter in fusion energy research. Each Server is implemented specifically to handle local data for an individual experiment. The Client chooses a specific server with which to link, but seldom needs to provide any other server-specific parameters. In rare circumstances, a particular server might require some special information (e.g., shot date, in installations where shot number is reset to '1' every day). The client would need to provide this special information in such a case.

3. How does local software "know" that data is remote ?

The obvious first question. Our design does not attempt to hide the location of the actual data, but to permit access to it through utility programs and subroutines with which the user is already familiar. The local library code might seek to make the data's location transparent to the user, but that is part of the library implementation, and is beyond the scope of this design.

In many cases, most of the data for a particular experiment will be at a remote site. For example, a physicist working from Oak Ridge on PBX data knows that the raw data, plus much analyzed data is in New Jersey. If some analyzed data signals are to be stored locally, then the local data-access library (e.g., DMG) might be designed to recognize this. Alternatively, some convention might be established whereby the user can indicate local or remote data.

MASTER

4. Basic concepts

The basic data-organization components which we define are

- the shot;
- the diagnostic; and
- the signal .

Each component has an associated "header" structure, although some fields of the structures may not be relevant to a particular client or server. Default values are defined for such fields. The individual components and the header information for each are described below.

In a future version, all data (including message headers) will be encoded in a computer-independent format by the sender and decoded by the receiver. At present, however, our "computer-independent" format is native-VAX.

Defined data types include REAL, CHAR, and signed and unsigned versions of INTEGER (4 bytes), WORD (2 bytes), and BYTE. There is also a special-purpose DATAGROUP format, used for describing dimension information. Other data types can be added if required.

We define a number of operations, performed by the server at the request of the client. The major operations are:

- OPEN_SHOT
Identifies the shot, possibly including "sub-shot" identification;
- OPEN_DIAGNOSTIC
Opens the particular diagnostic;
- GET_SIGNAL_HEADER
Returns information about specific signal, including "dimension" information (i.e., "timing").
- GET_SIGNAL_DATA
Returns specific data points, as requested by client.

Other defined operations are

- GET_SHOT_HEADER
Returns shot-header structure.
- GET_DIAG_HEADER
Returns diagnostic-header structure.
- CLOSE_DIAGNOSTIC; and
- CLOSE_SHOT.

4.1 The Shot

Shot identification is provided via an integer number and optional text field.

A "date" field is also provided, since some experiments reset their shot number to 1 at the beginning of each day. In such cases, identification is by date + shotnum.

Optional sub-shot information (integer number and text field) can also be specified. This will be useful for specifying calibration data, or on experiments with shots of long duration. In the latter case, a long pulse might be broken up into several "sub-shots". Some ATF diagnostics have already used the "sub-shot" concept, and it will become very common when ATF begins operation as a long-pulse experiment.

When we open a shot/subshot, the server returns a shot-id#. This is required when requesting shot header information, or when opening a Diagnostic.

The Shot Header includes such information as:

- shot date/time;
- list of available diagnostics (if possible).

4.2 The Diagnostic

A Diagnostic is identified by its name and a shot-id#. The server returns a diagnostic-id#, which uniquely identifies the Diagnostic for a particular shot. The diagnostic-id# is used when requesting the diagnostic's header information, and when requesting a specific signal.

Null Diagnostic: Some experimental machines may have no notion of individual "diagnostics", but rather have globally defined signal names. In these cases, the client should open a null diagnostic in order to get a proper diagnostic-id#.

The Diagnostic Header includes such information as:

- diagnostic date/time: if meaningful. File-creation time if diagnostics are stored in separate files;
- shot date/time. This seems redundant, but we might not have requested the shot header. Also, for some servers, the shot date/time may be available only after opening a diagnostic;
- list of available signals.

4.3 The Signal

Signals are identified by name and diagnostic-id#. (Remember, the diagnostic-id# uniquely identifies a diagnostic for a particular shot/subshot). There is no notion of a "signal-id#".

A particular signal can be further identified as being associated with (subordinate to) another signal, much as normal signals are associated with individual diagnostics. This allows us to support a file format, as exists on PBX, where a "main" signal (data channel) might have several symbols associated with it.

The Signal Header includes such information as:

- Signal date/time;
- Total number of points;
- Signal version number (if applicable);
- Quality factor (if applicable);
- Data information:
 - Format (floating pt, integers of various lengths);
 - "Compressed data" flag;
 - Compression algorithm id#. See under "Requesting Signal Data".
- Scale factors: arbitrary number of gains and offsets to be applied to data, at discretion of local library software. See below;
- Dimension information and values: describes the layout of the data. The protocol supports an arbitrary number of dimensions. See below;
- Comment of arbitrary length.

4.3.1 The Signal: Scale Factors

The client receives signal data in whatever format they are stored on disk. The signal header provides an arbitrary number of scale factors which can be applied to the signal data, at the discretion of the local library software.

Scale Factor information is

- gain (floating pt);
- offset (floating pt); and
- units (text).

The number of Scale Factors is provided in the signal header. Zero is a legitimate count -- it just means that the signal data are to be used as-is.

The main use of scale factors is the conversion of raw digitizer data (typically integers) to digitizer volts. A second scale factor might reflect the gain and offset provided by an amplifier. Other scale factors might be meaningful in certain situations.

When the signal header includes more than one scale factor, an "effective" scale factor is added, which provides the resultant gain and offset from all of the individual scale factors.

Analyzed data are most often stored on disk in their final format, and will likely have no associated scale factors.

4.3.2 The Signal: Dimension Information

The signal header includes "dimension information" which describes the arrangement of the signal's data. The protocol design supports an arbitrary number of dimensions.

Each dimension is described separately, in a format appropriate for the dimension:

- Random sampling interval: best described by an array of values, one for each element of the dimension's width. Values can be in any format which is appropriate;
- Fixed sampling interval (e.g., transient-digitizer data): Each sampling period is best described by providing a dataGroup specifying {start, delta, and count}. An arbitrary number of these dataGroups can be specified for the dimension. The dimension's width is the sum of the "count"s for all dataGroups.

In either case, the values for each dimension are provided in an array where the number of elements is specified in the signal header. The elements of the array may be in any of the defined data formats (real, integer, etc.), including the special "dataGroup" format used to describe data with fixed sampling interval.

5. Requesting Signal Data

After establishing a network link with the appropriate server, the client software (within the local library) requests a particular signal by sending a standard sequence of calls to the server :

- OPEN_SHOT: Client requests to open the shot: server returns shot-id#;
- OPEN_DIAGNOSTIC: Client requests to open the diagnostic for shot-id#: server returns diagnostic id#;
- GET_SIG_HEADER: Client requests the signal header. Signal header contains, among other things, the signal's "dimension" information (e.g., timing information). The shot header structure supports an arbitrary number of dimensions, with flexible formats for the actual dimension data.

- Client determines which point numbers it wants from each dimension in order to satisfy the user's request.
- GET_SIG_DATA: Client requests signal data, specifying starting point number and number of points for each dimension. Server returns the requested data.

Compressed data will be handled as follows <not yet implemented>:

- The system will, in time, define some standard data compression algorithms, assigning each a unique id#.
- If data are stored in compressed form, signal header will have "compressed data" flag set, and will include an algorithm id# if possible.
- Client, when requesting data, indicates whether or not it can accept the data in compressed form. If not, server will de-compress the data before shipping it to the client.

6. Network Communication

Our implementation uses the TCP/IP network protocol. Many other network standards would work as well, but TCP/IP is available where we want it.

In client-server interactions, each request initiated by the client generates a response from the server. The server always returns a status code in its response, telling the client whether its request was handled successfully.

Each message, in either direction, begins with a fixed-length message packet which includes a header and a data area. Message data comprises a data header, whose structure and length are specific to the operation being performed, and a variable-data area. The message header specifies the length of these two areas. The data header is guaranteed to fit within the initial message packet. If the variable-data portion will fit completely within the remaining space then it too is included within the initial message packet. Otherwise, the variable-data are passed in a succession of large-buffer transfers.

7. Design Considerations

It is worthwhile to mention a few of the specific criteria that guided our design, and some of the decisions that we made in implementing it.

- **FACT:** Home-grown, experiment-specific file formats are unavoidable in the real world. Even well-controlled data archives have the occasional exception: the one-dimensional signal which "everyone" knows is really two-dimensional, etc. Since servers are site-specific, it will know the peculiarities of the local data, and can serve it to clients in its correct form.
- **QUESTION:** Why not just NSF-mount the remote disks?
- **DECISION:** Refer to previous "FACT". In addition, the client-server arrangement was chosen because it:
 - a) minimizes the knowledge required of the client concerning where and how the data are stored at the server site; and
 - b) frees the server site to move and manage data at will.

- QUESTION: How should data be passed from server to client ?
- DECISION: Client receives signal data in the same format as it is stored in the data file. There are three reasons for this decision:
 - 1) Most raw data are stored in 2-byte integers, while scaled data is in floating-point format (4 bytes per value). So sending even uncompressed data puts a much smaller load on the network. If the data are compressed, the benefit will be even greater.
 - 2) Furthermore, integer data is probably easier to convert to and from a hardware-independent format, compared to floating-point values. And it is much more likely to compress well -- important for network loading.
 - 3) Finally, passing the rawest form of the data gives the client the flexibility to provide the user with scaled or unscaled data, and puts the work of scaling the data on the client's computer where it reasonably belongs.
- QUESTION: Who decides which data points should be returned?
- DECISION: The specification of the range of data requested, and the choice of data points to be returned, can be *very* specific to the local library. For example, some library routines expect a "from" time and a "to" time, while others expect a "from" time and a "number of points". Even the interpretation of which data point corresponds to the "from" time might be library-specific: is it the point at time greater-than or equal to requested time, the point less-than or equal to the requested time, or the point closest to the requested time?
 Our solution is to provide the client with detailed dimension information via the signal header. The client determines the specific data points to request, according to local conventions, and then requests only those points. The request mechanism is very flexible, and supports data stored with an arbitrary number of dimensions.

Appendix

DSDEF.H

This is the current state of the "include" file which contains all the parameters and data structures which we have defined so far.


```

/*****
 * DSDEF.H --
 *
 * Definitions of structures and parameters used in Data Server.
 *
 * Conventions:
 * - LENGTH fields: num bytes required for variable-length data. Always
 *   padded to modulo 4, ok? Use DS_TEXTBUFLen macro for text.
 *
 * History:
 * 23-Sep-1992 TRG Create. From discussions with Dave Greenwood (ORNL).
 *
 *****/

#ifndef DSDEF_LOADED
#define DSDEF_LOADED 1

/*****
 * Utility definitions and structures
 *****/

#define DS_TEXTBUFLen(nchars) ((nchars+sizeof(long)) & ~0x03)

/*****
 * ds_datetime: compact representation of date and time.
 *****/

typedef struct ds_datetime {
    short dssw_year; /* 0k for several thousand years. */
    char dssb_month; /* 1 - 12 */
    char dssb_day; /* 1 to 31 */
    char dssb_hours; /* 00 to 23 */
    char dssb_minutes; /* 00 to 59 */
    char dssb_seconds; /* 00 to 59 */
    char dssb_hundredths; /* 00 to 99 */
} DS_DATIME;

/*****
 * ds_dataGroup: DS-defined data type DS_DATA_GROUP,
 * used to pass dimension information.
 *****/

typedef struct ds_dataGroup {
    float vStart; /* Start value for group of data pts */
    float dv; /* Increment for each point */
    long count; /* Num pts in this dataGroup */
} DS_DATAGROUP;

/*****
 * data types
 *****/

#define DS_REAL 1 /* Floating pt number, 4 bytes */
#define DS_INTEGER 2 /* Signed integer, 4 bytes */
#define DS_WORD 3 /* Signed word, 2 bytes */
#define DS_BYTE 4 /* Signed byte, 1 byte */
#define DS_CHAR 5 /* ASCII character, 1 byte */
#define DS_UNSIGNED_INTEGER 6 /* Unsigned longword, 4 bytes */
#define DS_UNSIGNED_WORD 7 /* Unsigned word, 2 bytes */
#define DS_UNSIGNED_BYTE 8 /* Unsigned byte, 1 byte */
#define DS_DATA_GROUP 9 /* Indexing info: (struct ds_dataGroup) */

/*****

```

```

    * OPEN_SHOT:
    *=====
    /*-----
    * Request:
    *-----
    *-----

typedef struct ds_subshot {
    long sub$1_num; /* Shot/subshot number
    char sub$1_text[32]; /* ASCII text, null-terminated
    } DS_SUBSHOT;

typedef struct ds_openshot_request {
    long req$1_flags; /* Flags for read/write, etc
    #define DS_M_WRITE 0x01 /*..open for write
    long req$1_subshotCnt; /* Number of "subshot" structs
    long req$1_unassigned[10]; /* Shot date, if known
    struct ds_date req$1_shotdate; /* Primary shot number
    struct ds_subshot req$1_shotnum; /* Sub-shot info
    struct ds_subshot req$1_subshot[];
    } DS_OPENSOT_REQUEST;

/*-----
    * reply to OPEN_SHOT
    *-----
    *-----

typedef struct ds_openshot {
    long shot$1_shotid;
    } DS_OPENSOT;

/*=====
    * GET_SHOT_HDR:
    *=====
    *-----
    * request:
    *-----
    *-----

typedef struct ds_getshothdr_request { /* Shot id# from OPEN_SHOT
    long req$1_shotid;
    } DS_GETSHOTHDR_REQUEST;

/*-----
    * reply:
    *-----
    * ds_shothdr variable data:
    * "diaglist" is the first element in the "variableData"
    * portion of the shothdr. It is a string of diagnostic
    * names, each terminated by a single null byte.
    *-----

typedef struct ds_shothdr {
    long shot$1_shotno;
    struct ds_date shot$1_date;
    long shot$1_numDiagnostics;
    long shot$1_diaglistLen; /* num bytes required by diaglist
    long shot$1_unassigned[10]; /*..future expansion
    char shot$1_variableData[];
    } DS_SHOTHDR;

/*=====
    * OPEN_DIAGNOSTIC:
    *=====
    *-----
    * request:
    *-----
    *-----

typedef struct ds_opendiag_request {

```

```

* ds_msg: Message packet used to initiate any
* communication between client and server (in either
* direction).
* If "data" portion of communication is too big for
* the message packet, then ALL of the data is sent
* in succeeding data packet(s)....
=====
/*----- Fixed-length header for message packet.-----*/
* ds_msghdr: Fixed-length header for message packet.
/*-----*/

typedef struct ds_msghdr {
    char msgbb_majorRev; /* Major rev #: */
    char msgbb_minorRev; /* requested operation */
    short msgbb_callId;

#define DS_OPEN_SHOT 1
#define DS_GET_SHOT_HDR 2
#define DS_CLOSE_SHOT 3
#define DS_OPEN_DIAGNOSTIC 4
#define DS_GET_DIAG_HDR 5
#define DS_CLOSE_DIAGNOSTIC 6
#define DS_GET_SIG_HDR 7
#define DS_GET_SIG_DATA 8
    long msgbb_status; /* Status <from server> */
#define DS_SUCCESS 0 /* ...ok */
#define DS_NOSUCHSHOT 2 /* ...OPEN SHOT */
#define DS_NOSUCHDIAG 3 /* ...OPEN DIAGNOSTIC */
#define DS_BADSHOTID 4 /* ...bad shot id specified */
#define DS_NOSUCHSIG 5 /* ...GET SIG_HDR, GET_SIG_DATA */
#define DS_BADDIAGID 6 /* ...bad diagnostic id specified */
#define DS_BADDIM 7 /* ...GET SIG_DATA: bad "ndims" */
#define DS_DIMNOTVALID 8 /* ...illegal "dimension" request */
    long msgbb_hdrDataLen; /* Length (bytes) fixed data */
    long msgbb_varDataLen; /* Length (bytes) variable data */
} DS_MSGHDR;

/*----- The basic fixed-length message packet, initiates
* both the request from client-to-server and the response
* from server-to-client.
* Concerning DS_MSG_LEN: from "Networking with
* TCP/IP" by Douglas Comer, pp.68-69
* The Internet does not limit datagrams to a specific
* size, but suggests that networks and gateways should
* be prepared to handle datagrams of up to 576 octets
* (bytes) without fragmenting them.
*-----*/
#define DS_MSG_LEN 128 /* Length of basic message */
#define DS_MSGDATA_LEN (DS_MSG_LEN - sizeof(struct ds_msghdr))
#define DS_SENDRECVFLAGS 0

#define DS_MAXSENDSIZE (1024*4)
#define DS_SENDSIZE(x) (((x) > DS_MAXSENDSIZE) ? DS_MAXSENDSIZE : (x))

typedef struct ds_msg {
    struct ds_msghdr hdr; /* Message header */
    char msgdata[DS_MSGDATA_LEN]; /* Message data, if it all fits */
} DS_MSG;
/*-----*/

```

```

    long req$l_shotId;
    long req$l_flags;
    long req$l_diag_write 0x01
#define DS_M_DIAG_WRITE
    long req$l_diaglen;
    long req$l_unassigned[10];
    char req$t_diagnostic[];
    } DS_OPENDIAG_REQUEST;
}

/*-----
 * reply:
 *-----
typedef struct ds_open_diagnostic {
    long open$l_diagnosticId;
    } DS_OPEN_DIAGNOSTIC;
}

/*=====
 * GET DIAG HDR
 *=====
/*-----
 * request:
 *-----
typedef struct ds_getdiaghdr_request {
    long req$l_diagnosticId; /* from OPEN_DIAGNOSTIC */
    } DS_GETDIAGHDR_REQUEST;
}

/*-----
 * reply:
 * ds_diaghdr variable data:
 * "siglist" is the first element in the "variableData"
 * portion of the diaghdr. It is a string of signal
 * names, each terminated by a single null byte.
 *-----
typedef struct ds_diaghdr {
    struct ds_datetime diag$shotDateTime;
    /* note: there is no "diagnostic" datetime */
    long diag$l_numSignals;
    long diag$l_siglistlen;
    long diag$l_unassigned[10]; /* num bytes required by siglist */
    char diag$b_variableData[]; /* ..future expansion */
    } DS_DIAGHDR;
}

/*=====
 * GET SIG HDR
 *=====
/*-----
 * request:
 *-----
typedef struct ds_getsighdr_request {
    long req$l_diagnosticId; /* from OPEN_DIAGNOSTIC */
    char req$t_signalName[32];
    char req$t_associatedName[32];
    } DS_GETSIGHDR_REQUEST;
}

/*-----
 * reply:
 *-----
typedef struct ds_signal_header {
    char sig$t_name[32];
    struct ds_datetime sig$ datetime;
}

```

```

    long sig$1_npts;
    long sig$1_version;
    long sig$1_quality;
    long sig$1_format;
    long sig$1_flags;
#define DS_M_SIG_COMPRESSED 0x01
    long sig$1_cmprAlgo;
    long sig$1_scaleFactorCnt;
    long sig$1_scaleFactorLen;
    long sig$1_dimensionLen;
    long sig$1_dimensionalLen;
    long sig$1_commentLen;
    long sig$1_unassigned[10];
    char sig$b_variableData[];
} DS_SIGNAL_HEADER;

/* Total num pts for signal */
/* <opt> analyzed sig version num */
/* <opt> quality factor */
/* Format of signal data (e.g., REAL) */
/* Flags ... */
/* Data-compression id# */

long sig$1_npts;
long sig$1_version;
long sig$1_quality;
long sig$1_format;
long sig$1_flags;
long sig$1_cmprAlgo;
long sig$1_scaleFactorCnt;
long sig$1_scaleFactorLen;
long sig$1_dimensionLen;
long sig$1_dimensionalLen;
long sig$1_commentLen;
long sig$1_unassigned[10];
char sig$b_variableData[];
} DS_SIGNAL_HEADER;

/* Scaling factors: to be applied, if desired, to "raw"
 * signal data
 * If scaleFactorCnt is zero, data are used as-is.
 * If scaleFactorCnt is greater than zero, then an extra
 * "effective-scaleFactor" struct precedes the individual
 * scaleFactor structure(s). This is redundant for
 * the case where scaleFactorCnt=1, but that's life.
 */
typedef struct ds_scaleFactor {
    char units[12];
    float m;
    float b;
} DS_SCALEFACTOR;

/* Dimension information:
 */
typedef struct ds_perDimension {
    char dim$1_name[32];
    char dim$1_units[12];
    long dim$1_format;
    long dim$1_npts;
    long dim$1_unassigned[10];
    union {
        float dim$f_data[];
        long dim$l_data[];
        short dim$w_data[];
        char dim$b_data[];
    } u;
    struct ds_dataGroup dim$1_dataGroup[];
} DS_PERDIMENSION;

/* GET SIG_DATA
 */
/* request:
 * "ds_dataRequest" appears inside ds_getsigdata_request.
 */
typedef struct ds_dataRequest {
    long first;
    long npts;
} DS_DATAREQUEST;

```

```

typedef struct ds_getsigdata_request { /* from OPEN_DIAGNOSTIC */
    long req$1_diagnosticId;
    char req$1_signalName[32];
    char req$1_associatedName[32];
    long req$1_flags;
    #define DS_M_OK_COMPRESSED 0x01 /*..ok to send compressed data */
    long req$1_ndims; /* Num dimensions in dataRequest */
    long req$1_unassigned[10];
    struct ds_dataRequest req$1_dataRequest[];
} DS_GETSIGDATA_REQUEST;

/*-----
 * reply:
 *-----*/

typedef struct ds_signal_data {
    long data$1_npts;
    long data$1_format; /* Format of data as returned */
    long data$1_flags; /* Flags ... */
    long data$1_cmpr$Algorithm;
    long data$1_unassigned[2];
    char data$1_variableData[]; /* the actual data */
} DS_SIGNAL_DATA;

/*=====
 * CLOSE_SHOT:
 *=====*/

typedef struct ds_close_shot {
    long close$1_shotId;
} DS_CLOSE_SHOT;

/*=====
 * CLOSE_DIAGNOSTIC:
 *=====*/

typedef struct ds_close_diagnostic {
    long close$1_diagnosticId;
} DS_CLOSE_DIAGNOSTIC;

#endif /* End of DSDEF.H */

```

EXTERNAL DISTRIBUTION IN ADDITION TO UC-420

Dr. F. Paoloni, Univ. of Wollongong, AUSTRALIA	Prof. I. Kawakami, Hiroshima Univ., JAPAN
Prof. M.H. Brennan, Univ. of Sydney, AUSTRALIA	Prof. K. Nishikawa, Hiroshima Univ., JAPAN
Plasma Research Lab., Australian Nat. Univ., AUSTRALIA	Director, Japan Atomic Energy Research Inst., JAPAN
Prof. I.R. Jones, Flinders Univ, AUSTRALIA	Prof. S. Itoh, Kyushu Univ., JAPAN
Prof. F. Cap, Inst. for Theoretical Physics, AUSTRIA	Research Info. Ctr., National Inst. for Fusion Science, JAPAN
Prof. M. Heindler, Institut für Theoretische Physik, AUSTRIA	Prof. S. Tanaka, Kyoto Univ., JAPAN
Prof. M. Goossens, Astronomisch Instituut, BELGIUM	Library, Kyoto Univ., JAPAN
Ecole Royale Militaire, Lab. de Phy. Plasmas, BELGIUM	Prof. N. Inoue, Univ. of Tokyo, JAPAN
Commission-European, DG. XII-Fusion Prog., BELGIUM	Secretary, Plasma Section, Electrotechnical Lab., JAPAN
Prof. R. Bouciqué, Rijksuniversiteit Gent, BELGIUM	S. Mori, Technical Advisor, JAERI, JAPAN
Dr. P.H. Sakanaka, Instituto Fisica, BRAZIL	Dr. O. Mitarai, Kumamoto Inst. of Technology, JAPAN
Instituto Nacional De Pesquisas Especiais-INPE, BRAZIL	J. Hyeon-Sook, Korea Atomic Energy Research Inst., KOREA
Documents Office, Atomic Energy of Canada Ltd., CANADA	D.I. Choi, The Korea Adv. Inst. of Sci. & Tech., KOREA
Dr. M.P. Bachynski, MPB Technologies, Inc., CANADA	Prof. B.S. Liley, Univ. of Waikato, NEW ZEALAND
Dr. H.M. Skarsgard, Univ. of Saskatchewan, CANADA	Inst of Physics, Chinese Acad Sci PEOPLE'S REP. OF CHINA
Prof. J. Teichmann, Univ. of Montreal, CANADA	Library, Inst. of Plasma Physics, PEOPLE'S REP. OF CHINA
Prof. S.R. Sreenivasan, Univ. of Calgary, CANADA	Tsinghua Univ. Library, PEOPLE'S REPUBLIC OF CHINA
Prof. T.W. Johnston, INRS-Energie, CANADA	Z. Li, S.W. Inst Physics, PEOPLE'S REPUBLIC OF CHINA
Dr. R. Bolton, Centre canadien de fusion magnétique, CANADA	Prof. J.A.C. Cabral, Instituto Superior Tecnico, PORTUGAL
Dr. C.R. James., Univ. of Alberta, CANADA	Dr. O. Petrus, AL I CUZA Univ., ROMANIA
Dr. P. Lukác, Komenského Univerzita, CZECHO-SLOVAKIA	Dr. J. de Villiers, Fusion Studies, AEC, S. AFRICA
The Librarian, Culham Laboratory, ENGLAND	Prof. M.A. Hellberg, Univ. of Natal, S. AFRICA
Library, R61, Rutherford Appleton Laboratory, ENGLAND	Prof. D.E. Kim, Pohang Inst. of Sci. & Tech., SO. KOREA
Mrs. S.A. Hutchinson, JET Library, ENGLAND	Prof. C.I.E.M.A.T, Fusion Division Library, SPAIN
Dr. S.C. Sharma, Univ. of South Pacific, FIJI ISLANDS	Dr. L. Stenflo, Univ. of UMEA, SWEDEN
P. Mähönen, Univ. of Helsinki, FINLAND	Library, Royal Inst. of Technology, SWEDEN
Prof. M.N. Bussac, Ecole Polytechnique., FRANCE	Prof. H. Wilhelmson, Chalmers Univ. of Tech., SWEDEN
C. Moutet, Lab. de Physique des Milieux Ionisés, FRANCE	Centre Phys. Des Plasmas, Ecole Polytech, SWITZERLAND
J. Radet, CEN/CADARACHE - Bat 506, FRANCE	Bibliotheek, Inst. Voor Plasma-Fysica, THE NETHERLANDS
Prof. E. Economou, Univ. of Crete, GREECE	Asst. Prof. Dr. S. Cakir, Middle East Tech. Univ., TURKEY
Ms. C. Rinni, Univ. of Ioannina, GREECE	Dr. V.A. Glukhikh, Sci. Res. Inst. Electrophys. Apparatus, USSR
Dr. T. Mui, Academy Bibliographic Ser., HONG KONG	Dr. D.D. Ryutov, Siberian Branch of Academy of Sci., USSR
Preprint Library, Hungarian Academy of Sci., HUNGARY	Dr. G.A. Eliseev, I.V. Kurchatov Inst, USSR
Dr. B. DasGupta, Saha Inst. of Nuclear Physics, INDIA	Librarian, The Ukr.SSR Academy of Sciences, USSR
Dr. P. Kaw, Inst. for Plasma Research, INDIA	Dr. L.M. Kovrizhnykh, Inst. of General Physics, USSR
Dr. P. Rosenau, Israel Inst. of Technology, ISRAEL	Kernforschungsanlage GmbH, Zentralbibliothek, W. GERMANY
Librarian, International Center for Theo Physics, ITALY	Bibliothek, Inst. Für Plasmaforschung, W. GERMANY
Miss C. De Palo, Associazione EURATOM-ENEA, ITALY	Prof. K. Schindler, Ruhr-Universität Bochum, W. GERMANY
G. Grosso, Istituto di Fisica del Plasma, ITALY	Dr. F. Wagner, (ASDEX), Max-Planck-Institut, W. GERMANY
Prof. G. Rostangni, Istituto Gas Ionizzati Del Cnr, ITALY	Librarian, Max-Planck-Institut, W. GERMANY
Dr. H. Yamato, Toshiba Res & Devel Center, JAPAN	Prof. R.K. Janev, Inst. of Physics, YUGOSLAVIA

END

**DATE
FILMED**

2/12/93