

# Tunnel-Data Server: Data format, archive maintenance and access protocol.

Mechanical Engineering Report 2003/06

P. A. Jacobs

Centre for Hypersonics  
The University of Queensland.

September 25, 2003

## **Abstract**

The Tunnel-Data server provides access to the archive of transient data that has been collected by the Hypersonics Group over the past 20 years. These data are associated with thousands of individual shots of the shock tunnel and expansion tube facilities run by the group and are made available via a web server using a simple text-based protocol. This report describes the layout of the data archive, the format of the individual files and the details of the protocol used to access the data. It also describes the scripts used to maintain the archive files, serve data from the archive via CGI, and provide the client-side access for data-analysis environments such as MATLAB and Octave. Much of the detailed documentation is embedded in these scripts.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Archive Structure and Data Format</b>	<b>5</b>
<b>3</b>	<b>Maintenance of the Archive</b>	<b>8</b>
3.1	Setting up the server CGI script . . . . .	9
<b>4</b>	<b>Data Access Protocol</b>	<b>10</b>
<b>5</b>	<b>Concluding Remarks</b>	<b>13</b>
<b>A</b>	<b>Server Script</b>	<b>15</b>
A.1	td_server.tcl . . . . .	15
<b>B</b>	<b>Archive Maintenance Scripts</b>	<b>23</b>
B.1	td_scan.tcl . . . . .	23
B.2	update_monc_files.tcl . . . . .	28
<b>C</b>	<b>MATLAB/Octave Client Scripts</b>	<b>30</b>
C.1	Web Access via an External Process . . . . .	30
C.1.1	get_data.m . . . . .	30
C.1.2	td_web_client.tcl . . . . .	32
C.1.3	test_data_server.m . . . . .	33
C.2	Java-based Network Access . . . . .	35
C.2.1	fetch_text_from_server.m . . . . .	35
C.2.2	fetch_channel_data.m . . . . .	36
C.2.3	fetch_channel_header.m . . . . .	36
C.2.4	demonstrate_fetch.m . . . . .	36

# 1 Introduction

The Hypersonics Group at the University of Queensland has been operating a set of shock tunnels and expansion tubes for approximately twenty years. There are many thousands of data files collected from about 10000 shots, with most data (at UQ) being stored in a binary format by MONC<sup>1</sup>, a specialized data-acquisition program written in Turbo-Pascal. With each passing year, there are to be fewer computer models that run the MONC program. Its integrated graphical display is based on the Borland Graphics Interface to EGA graphics cards that seem to be not always properly supported in current hardware. MONC has served us well and will continue to be the main data-acquisition program for the near future, however, its data display capability needs to be replaced.

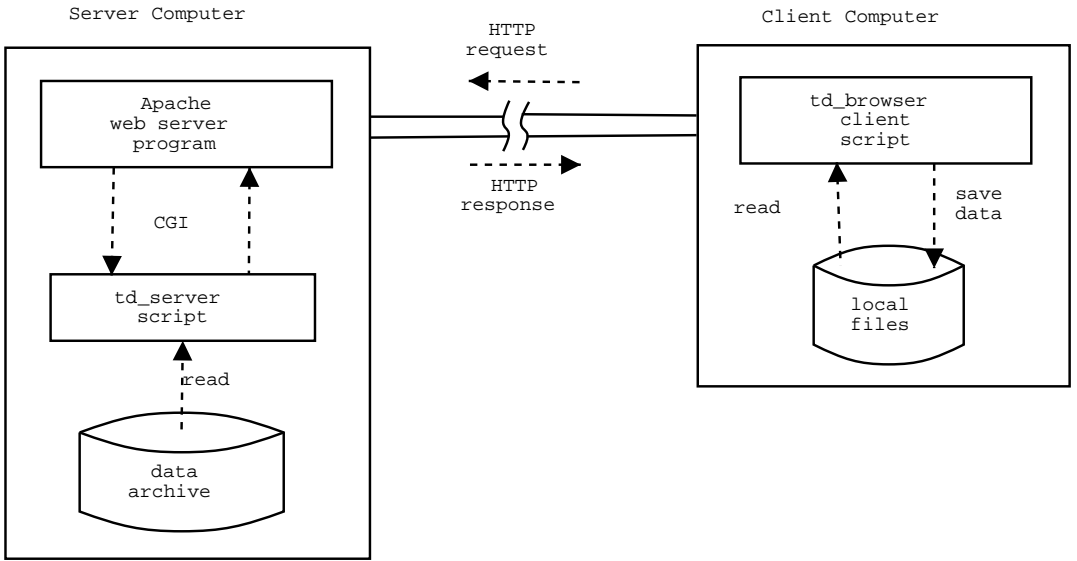
Another consideration is that with the Centre now involving academic groups and their experimental facilities across the country, there is a need to make the data from each individual facility available to all members of the Centre, irrespective of location and computing facilities.

Although there are sophisticated data server projects (see e.g. [1, 2]) that provide access to and subsetting of extremely large data sets, they are much more complicated than we would like. The approach that we have taken is to store the sensor data in text files that have been compressed with widely-available (GNU) software and then make the reconstituted data available via a CGI script invoked by a simple text request to a web server. This client-server architecture is shown in Figure 1. The hypertext transport protocol is reliable and provides a simple way to connect a client computer with a remote data archive. The text and image transport mechanisms of a standard web server (such as Apache [3]) are all that are required for the distribution of our data and, as a bonus, moderate security is provided by the Basic Authentication capability of the web server.

Maintenance of the data archive involves a conversion from several binary formats to a plain text format that should be suitable for many years to come. Although it is convenient to retain the directory-naming hierarchy used by MONC, the hierarchy is extended to include a number of experimental facilities. Important changes to the storage arrangement are a move to one file per channel (or sensor) and the storage of the metadata at the top of each channel file. This allows data from other facilities such as the HEG shock tunnel in Göttingen to be easily converted and stored in the same format.

---

<sup>1</sup>The MONC program was originally written by Richard Morgan for an Apple IIe connected to a Biomation Transient recorder. It was later rewritten by Allan Paull for an IBM-PC connected to another data-acquisition system provided by NASA Langley and eventually adapted to the locally developed transient recorders that are used today.



HTTP Request: Start td\_server (CGI) script  
with query string  
"facility=T4+shot=7319+channel=110+part=data"

HTTP Response:

```
Content_type: text/plain
0.00000e+000 4.02832e+001
2.00000e+000 -8.54492e+000
4.00000e+000 -8.54492e+000
6.00000e+000 1.58691e+001
8.00000e+000 4.02832e+001
...
```

Figure 1: Client-server architecture with an example of a hypertext request from the client and response from the server.

The following sections provide the details of the new data format, the client-server architecture of the new data system and the scripts that are used to maintain and access the archive. A companion report [4] is available online. It is a hypertext document containing the machine-readable form of the Tunnel-Data Server software collection. This collection includes a client browser as well as the server-side scripts discussed in the present report.

## 2 Archive Structure and Data Format

As done for the original MONC program, we use the standard file system to provide a hierarchical structure to the data collection. The toplevel directory contains one directory for each facility (*e.g.* T4, X2 and X3) and, within the facility directory, there is one directory for each shot and, within that shot directory, one data file for each recorded channel. The structure of the archive is

```

moncdata + T4 + rundesc + 7319.txt      (original text description for shot)
          |           |           - ...
          |           |
          |           + descript + 7319.mod  (original MONC file with model data)
          |           |           - ...
          |           |
          |           + 7319 + 7319.011      (original MONC data file)
          |           |           - ...
          |           |           - 7319A.LST.gz  (new list of channel numbers and names)
          |           |           - 7319A.110.gz  (new data file)
          |           |           - ...
          |           + ...                  (other T4 shots)
          |
          + X3 + ...
          |
          + X2 + ...

```

The new data file names use the shot name with an appended “A” to distinguish the current (ASCII) data files from the original (binary) MONC files. The extension is an integer value made up of digits representing: the card in the databox (1–7), the analog-to-digital converter channel (1, 2, 3), and the subchannel in a multiplexed signal (1, 2, 3, 4). A digit 0 for the subchannel indicates that the data was not multiplexed. The new text files are much larger than the original MONC files but they are stored on

the server in “gzipped” form which reduces their size considerably. The actual file name for the present data, in compressed form, is then “7319A.110.gz”. The text for the run description is left as-is in the “rundesc” directory. It is contained in the file “7319.txt”. Note that the original MONC files are kept in the same directories and are not altered by the current system.

Although the proposed server and client arrangement for accessing the data across the internet relieves individual experimenters of having to store and organise their own data sets, one can never be too paranoid. Having one channel per file and all files for an experiment in the one directory should make it easy to backup or transport individual shots manually.

It is easiest to explain the new format by example. The top 40 lines of the uncompressed (ASCII) data file 7319A.110 are:

```
# dataSource MONC v4.8, extracted by defrock
# shotName 7319
# channelId 110
# withTimeColumn no
# dataPoints 8192
# dataType scaled
# dataUnits kPa
# timeStart 0.00000E+0000
# timeAverageWindow 0.00000E+0000
# timeInterval 2.00000E+0000
# timeUnits microseconds
# transducerSensitivity 1.00000E-0004
# transducerSensitivityUnits volts/kPa
# transducerName spa
# transducerLocation 0.00000E+0000
# transducerSerialNumber 4266
# gain 1.00000E+0000
# qfluxgain 1.00000E+0000

4.02832E+0001
-8.54492E+0000
-8.54492E+0000
1.58691E+0001
4.02832E+0001
-8.54492E+0000
-8.54492E+0000
-8.54492E+0000
-8.54492E+0000
-8.54492E+0000
-8.54492E+0000
-8.54492E+0000
4.02832E+0001
-8.54492E+0000
-8.54492E+0000
-8.54492E+0000
-8.54492E+0000
-8.54492E+0000
-8.54492E+0000
-8.54492E+0000
-8.54492E+0000
-8.54492E+0000
4.02832E+0001
```

The content of the new data file is in two parts: first some metadata then, the actual sampled data, scaled to appropriate physical units and represented by floating-point numbers in exponential format. There is a blank line between the header and the data body. This format is easy to read and easy to parse automatically in almost any programming language. It has the added convenience of being able to be read directly by GNUPlot. Porting the entire data collection to a new format, if we so wish, should be trivial.

This particular file has been written by the data conversion program, defrock, which is a stripped-down version of the current MONC program. Although defrock is not detailed here, its source is included in the online software collection[4]. It is a command-line program (on both MS-DOS and Linux systems) that can be invoked to convert the data for a single shot. The scripts described later can automate the process of locating and converting many shots.

Lines, at the top of the file, that begin with the sharp (or hash) character are metadata which describe the source of the following numerical data and give some indication of what physical quantities that data represent. The metadata elements consist of name-value pairs and are based on the items recorded by the MONC program. It may be advantageous to vary the specification as the need arises. The data server and client should be flexible enough to work with whatever elements are available, with sensible default values being assumed for missing elements. With the metadata now stored in the same file as the actual data, we have eliminated the confusion and uncertainty that accompanies binary data sets with separate description files.

This particular file (for T4 shot 7319) contains only the sampled data for a pressure transducer – actually, the stagnation pressure transducer A located at the end of the shock tube. As indicated by the metadata, there are no time stamps for individual samples; they were all taken with the fixed sample period of  $2\mu s$ . There is enough information to reconstruct the sample times relative to an arbitrary starting time which is presumed to be the same for all channels. When the server is asked for some data, it first reconstructs the time stamp for each sampled value, and then sends a sequence of time-value pairs to the client.

Most of the T4 and X-tube data has been recorded with a single sample period for each channel but the data-acquisition hardware is capable of changing sample period part way through a recording. For cases where the sample period has changed during the recording, individual time-stamps would be put in the first column of the file with the sampled data values placed in the second column. The “withTimeColumn” element in the header would then have a value of “yes”. Most of the HEG data recordings seem to

be of this form.

### 3 Maintenance of the Archive

The T4 data archive as seen by a client computer is located on a server computer (`proba.uq.edu.au`) that is completely separate from the data-acquisition computers in the laboratory. It is the daily responsibility of individual experimenters to make sure that valid copies of their data files are put in the correct locations on the Novell backup drive (R:). It is from these files that the T4, X2 and X3 archives are made.

Scripts to automate the process of copying, converting, and cataloging the data are given in Appendix B. Presently, this process starts with data files that have been copied from the Novell backup drive (R:) onto an intermediate Linux workstation. On my workstation, with the Netware client package installed, the whole process is started with the specific commands:

---

```
$ sudo ncpmount -S uqspace -U peterj -b /mnt/uqspace
**** password required for Novell system ****
$ cd ~/cfd/tds/server/
$ ./update_monc_files.sh
... lots of text written indicating the progress of the script ...
$
```

---

where the R: drive has been mounted as `/mnt/uqspace` and `update_monc_files` shell (see Appendix B) script coordinates the copying of the MONC files from the R: drive, scanning and converting valid shots, and then copying all of the files to the web-server. To keep the archive current, this process should be performed each day. As part of the update process, the Tcl script `td_scan.tcl` is used to search for complete shots and then generate associated text data files by invoking the `defrock` program. The whole process of updating the collection may take several minutes and a complete build from scratch may take (a small number of) hours. The final stage of making a copy of all of the files on the web server uses secure copy (`scp`) so, for the convenience of being able to let the script proceed without user intervention, passwords for `scp` access to the web server can be replaced by the use of a pair of `ssh` keys.

Staging the processing across three machines provides some confidence that the scanning of the MONC files on the intermediate workstation does not damage any of the

master files on the Novell system. Having the web server separate again allows easy replacement of the publicly accessible files in case of accidents or deliberate damage to the web-server's file system.

Backing up of the whole archive from the intermediate workstation is conveniently done with GNU tar and a CD-R writer. The entire archive can be written to a set of tape-archive files with the command:

```
$ tar --tape-size 700000 -cvf T4_data.tar moncdata/T4/
$
```

Each time the tar file reaches the specified size, it is closed and a prompt appears indicating that the “tape” is full. From another window, rename the file just written:

```
$ mv T4_data.tar T4_data_tape1.tar
$
```

The closing of the tar file at a set size will most probably result in truncating the last file added. A complete copy of the same file will be the first file added to the next tar file written. The size specified above fits onto a 700Mb CD-R with a bit of space to spare. The tar files so recorded are readable on Linux/UNIX, MS-Windows and Macintosh computers.

### 3.1 Setting up the server CGI script

The program that actually responds to your request for data is a Tcl script (`td_server.tcl`, see Appendix A) in the directory `/var/www/cgi-bin/tds/`. This program is invoked by the web server if the request has come from an authorised user as specified in the “.htaccess” file accompanying the server script:

```
# .htaccess file for td_server on RedHat 8.0
# installed as /var/www/cgi-bin/tds/.htaccess

AuthName "tds"
AuthType Basic
AuthUserFile /etc/httpd/conf/users

require user tdguest peterj
```

---

The contents of the authorised user and password file might be:

---

```
tdguest :YhsrZoKHKdGco
peterj :HdVgvlmyT3LUA
```

---

where the “tdguest” entry could have been added with the command:

---

```
sudo htpasswd -b /etc/httpd/conf/users tdguest tdpaswd
```

---

Note that this username and password is the default in the Tunnel Data Browser, as distributed, and that the only data that will be served to this identity is a very restricted set from the T4 collection. To get access to other parts of the archive, you will need another (authorised) username and password.

## 4 Data Access Protocol

As shown in Figure 1, access to the data is via a Common Gateway Interface (CGI) request [5] to a standard web server. We use the Apache web server [3] to invoke the server script `td_server.tcl` (see Appendix A) to select the data from the archive. When `td_server.tcl` is invoked by the web server, it gets its request from the text in the `QUERY_STRING` environment variable. The script then parses this string to determine what particular data has been requested. If that data is available, it uncompresses the appropriate file and sends the requested data to the web server as plain text. The server script is written in the Tool Control Language (Tcl) [6] which is freely available for all of our computer systems. Although it was originally written as a prototype, `td_server.tcl` seems to run fast enough to handle our group’s current needs, even on old hardware.

The protocol is text-based and you may request data manually via a standard web browser by entering the URL for the server script with the query string attached. For example, on accessing the URL

---

```
http://www.mech.uq.edu.au/cgi-bin/td_server.tcl?
↔facility=T4+shot=7319+channel=110+part=data
```

---

the system should first challenge you for a valid username and password and then should result in the return of the time history of the nozzle-supply pressure transducer A in shot 7319 of the T4 shock tunnel. The ↔ indicates that the command should be entered

without the line-break that appears here. The history data are returned as a stream of text, with two numerical values (time and pressure) per line. If the time values are not already stored in the ASCII data file, they are reconstructed from the metadata by the server script, assuming a fixed sample period.

To get the metadata for the channel, the URL

```
http://www.mech.uq.edu.au/cgi-bin/td_server.tcl?  
↔facility=T4+shot=7319+channel=110+part=info
```

will return the descriptive elements from the head of the data file. These are returned as a text stream with one name-value pair per line. The sharp (or hash) characters seen in the sample file (Section 2) are not included.

For further options, such as requesting a list of shots or list of channels for a particular shot, look at the introductory comments in the `td_server.tcl` script.

Appendix C contains MATLAB [7] and Octave [8] functions that allow the user to get the tunnel data into those data-analysis environments. There are two separate sets of client files. In the first set, the `get_data.m` file contains a function that uses an external program (`wget`) or script (`td_client.tcl`) to talk to the web server. If the program `wget` is available on your workstation, use it in preference to the `td_web_client.tcl` script because, without having to start up the Tcl interpreter, it should involve less computational overhead.

The example coordination script, `test_data_server.m` was used to capture data with `wget` and generate the plot in Fig. 2. The essential lines are:

```
facility = 'T4';  
shot_id = '7319';  
channel = '110';  
[attrib, value] = get_data( facility, shot_id, channel, 'info' );  
[t, v] = get_data( facility, shot_id, channel, 'data' );
```

The first call collects the name-value pairs for the metadata at the start of the file and the second call collects the actual sensor history data. Note that all of the arguments to the function are strings. The rest of the script deals with the optional use of the Java-based access functions, data filtering and plotting differences between MATLAB and Octave. Once the data is collected, the test script proceeds to filter and plot the data.

The same coordination script can be used to access the data via the Java VM that

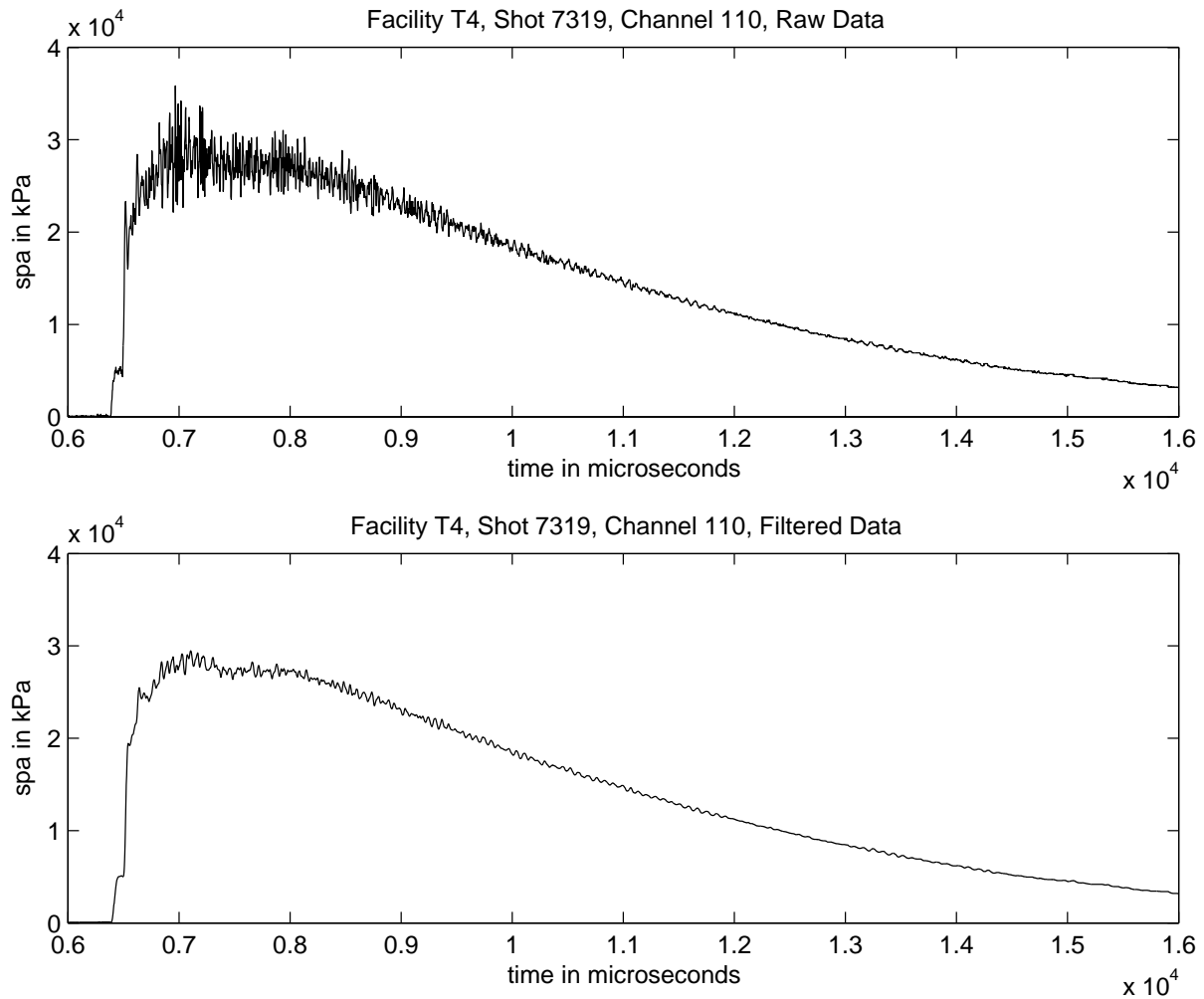


Figure 2: Plot produced by the test\_data\_server.m script in Appendix C.

comes as part of recent a version of MATLAB, however, performance of these functions was disappointing. On the test client and server, running on a 10 Mbit/s network, the Java-based functions required 38 seconds to fetch the data. This compares badly with the 2 seconds required for the wget-based access functions.

Of course, if you just want the data saved to a local file, it is easy to use the wget to do so with the command:

```
wget --output-document=my_file.data \  
      --http-user=tdguest \  
      --http-passwd=tdpasswd \  
      --quiet \  
http://www.mech.uq.edu.au/cgi-bin/td_server.tcl?\  
facility=T4+shot=7319+channel=110+part=data
```

Also, if you don't wish to mess with this data access protocol directly, there is a stand-alone data browser [4] written in Tcl/Tk and available for use with both MS-Windows and Linux workstations.

## 5 Concluding Remarks

The current server and client system is a proof-of-concept prototype. Although it makes the data from individual channels easily accessible from any desktop that has internet access, further work is yet to be done to collect and make available the experiment meta-data. This is presently only available by browsing the individual run-description text files but it could be made available for searches via a database.

Other jobs that need attention are: (1) the use of a compressed format to transport of the data across the internet and (2) the inclusion of other data such as the flow images collected for some shots. A reimplementaion of the server script in a web-specific language such as PHP [9] would possibly make the script shorter and also make the long-term maintenance easier.

## References

- [1] Distributed Oceanographic Data System – DODS. URL, <http://www.unidata.ucar.edu/packages/dods/>.
- [2] GrADS-DODS Server (GDS). URL, <http://www.grads.iges.org/grads/gds/>.
- [3] Apache web server. URL, <http://www.apache.org/>.
- [4] P. A. Jacobs. TDS - shock tunnel data server and browser. Department of Mechanical Engineering Report 2002/08, The University of Queensland, Brisbane, Australia., January 2002. URL <http://www.mech.uq.edu.au/staff/jacobs/cfcd/>.
- [5] D. Maggiano. *CGI programming with Tcl*. Addison Wesley Longman, Reading, Massachusetts, 2000.
- [6] B. B. Welch. *Practical Programming in Tcl and Tk*. Prentice Hall PTR, Upper Saddle River, NJ, 2000.
- [7] MATLAB. URL, <http://www.mathworks.com/>.
- [8] Octave. URL, <http://octave.sourceforge.net/>.
- [9] PHP home page. URL, <http://www.php.net/>.

# A Server Script

## A.1 td\_server.tcl

```
#!/bin/sh
#\
exec /usr/local/ActiveTcl/bin/tclsh "$0" "${1+"$@"}"

# File:
# td_server.tcl
#
# Purpose:
# Send some shock tunnel data via the CGI interface.
#
# Command-line Usage:
# td_server.tcl facility=<facility>+shot=<shot>+channel=<channel>+part=<part>
# where:
# <facility> is one of (T4, X1, X2, X3)
# <shot> is the part of the file name that identifies the shot.
#     MONC data (and the equivalent compressed ASCII data) is stored in
#     a directory of this name.
#     If a special value of "list" is supplied, a list of shots for
#     the specified facility is returned.
# <channel> is the channel identifier, also used by MONC as the extension
#     of the channel's data file.
#     A value of "list" will cause the server to return a list of channel
#     numbers and transducer names, one channel per line.
#     A value of "rundescription" will cause the server to return the
#     content of the run description file.
# <part> is one of (info, data, raw)
#     A value of "info" causes the server to return the header information
#     for the channel.
#     A value of "data" will cause the server to return just the history
#     data as a number of "time value" pairs.
#     A value of "raw" will cause the server to return the uncompressed
#     data file unprocessed.
#     This parameter is optional as it has no meaning if a particular
#     channel is not specified and it will default to a value of "data"
#     if a valid channel is specified.
#
# CGI Usage:
# Instead of getting the query string from the command-line,
# the script looks in the environment variable QUERY_STRING.
#
# Author:
# Peter Jacobs
# Department of Mechanical Engineering
# The University of Queensland
#
# Revisions:
# 26-Dec-01 original code
# 29-Dec-01 CGI working, times generation
# 30-Dec-01 documentation, part selector, send list of channels
# 31-Dec-01 options to return shot list and run descriptions
# 01-Jan-02 generalize for Win32 use
# 05-Jan-02 include keyword "td_server_error" in returned error messages
#     return the raw/original data file if requested
# 29-Mar-02 put all results into a content string so that the size
#     may be sent as part of the HTTP header.
# 01-Jun-03 Added allowAccess procedure to filter CGI requests.

#----- procedures -----

proc initializeData {} {
    global td contentString allowedShotList
    set contentString ""

    if { [string equal $::tcl_platform(platform) windows] } {
```

```

    # Home directory for the MONC data
    set td(rootDir) [file join d: /home moncdata]
    # A temporary file for decompressing the MONC data files, etc.
    set td(tempDataFile) [file join d: /tdsTempData]
} else {
    # Home directory for the MONC data; there is more than one possibility.
    set td(rootDir) [file join / home moncdata]
    if { ![file exists $td(rootDir)] } {
        set td(rootDir) [file join / home2 moncdata]
    }
    # A temporary file for decompressing the MONC data files, etc.
    set td(tempDataFile) [file join / tmp tdsTempData]
};
# Try to make the temporary data file name unique.
append td(tempDataFile) [expr int(rand() * 100000000)]

set td(dataDir) ""
set td(dataFile) ""

# check if we are in a CGI script environment
set td(cgiScript) [info exists ::env(QUERY_STRING)]

# Data for the allowAccess procedure: for each restricted user,
# there is a list of facility+shotId pairs.
# We may put in a few sample sets but we'll need to think
# of something "scalable" it it gets out of hand.
set allowedShotList(tdguest) [list T4+7319 T4+7320 T4+list X3+list X2+list]
}; # end proc

proc sendContentString {} {
    global td
    global contentString

    if { $td(cgiScript) } {
        puts "Content-type: text/plain"
        puts "Content-length: [string length $contentString]"
        puts ""
    }; # end if
    puts -nonewline $contentString
}; # end proc

proc parseQueryString {} {
    # We may have started this script from the normal shell and
    # have passed the query string via the command line.
    # First, check the QUERY_STRING environment variable and,
    # if there is nothing there, try the command-line.

    global td

    # default values for the expected command-line parameters
    set td(facility) ""
    set td(shot) ""
    set td(channel) ""
    set td(part) "data"

    if { [info exists ::env(QUERY_STRING)] } {
        set queryString $::env(QUERY_STRING)
    } else {
        set queryString [join $::argv "+"]
    }; # end if

    foreach {a} [split $queryString "+"] {
        foreach {name value} [split $a "="] {
            set td($name) $value
            # puts "name: $name, value: [set td($name)]"
        }; # end foreach
    }
}; # end proc

```

```

    }; # end foreach
}; # end proc

proc sendTestData {} {
    # Generate some dummy data
    global contentString
    for {set t 0.0} {$t < 10.0} {set t [expr $t + 0.1]} {
        set v [expr sin($t)]
        append contentString "$t $v\n"
    }; # end for
}; # end proc

proc looksLikeAShotDir { shotName } {
    # From the main data directory, look for the gzipped channel list file
    # within the shot directory.
    set pattern [join [list $shotName / $shotName A . {{LST, lst}} .gz] "" ]
    set listFileList [glob -nocomplain $pattern]
    if { [llength $listFileList] > 0 } {
        set result 1
    } else {
        set result 0
    }; # end if
    return $result
}; # end proc

proc buildShotList {} {
    # Look up the facility directory and extract a list of shot directories.
    # Return the list as a string, with several shot names per line.
    global td

    set saveDir [pwd]
    set dataDir [file join $td(rootDir) $td(facility)]
    if { [file isdirectory $dataDir] } {
        cd $dataDir

        if { [file exists "shot.index"] } {
            # Read the shot list from the file.
            set indexFile [open "shot.index" r]
            set shotList [read -nonewline $indexFile]
            close $indexFile
            # puts "shot.index content: $shotList ."
            set infoSource "(used shot.index file)"
        } else {
            # Build the shot list by searching individual directories.
            set shotList [glob -nocomplain *]
            # puts "In directory [pwd], before filter, shotList is $shotList"
            # Filter out those directories that appear
            # to NOT contain shot data.
            foreach dirName $shotList {
                if { ![looksLikeAShotDir $dirName] } {
                    set i [lsearch $shotList $dirName]
                    if { $i >= 0 } {
                        set shotList [lreplace $shotList $i $i]
                    }; # end if
                }; # end if
            }; # end foreach
            # puts "After filter, shotList is $shotList"
            set infoSource "(searched directories)"
        }; # end if
        cd $saveDir
        # We actually want the end result with one shot per line.
        set resultString "For facility $td(facility), there are "
        append resultString "[llength $shotList] shots with ASCII data."
        append resultString " $infoSource \n"
        foreach {s0 s1 s2 s3 s4 s5 s6 s7 s8 s9} $shotList {

```

```

        append resultString "$s0 $s1 $s2 $s3 $s4 $s5 $s6 $s7 $s8 $s9 \n"
    }; # end foreach
} else {
    set resultString "td_server_error : $dataDir is not a directory."
}; # end if
return $resultString
}; # end proc

proc sendListOfShots {} {
    # This new version used the same procedure as td_browser.
    global contentString
    append contentString [buildShotList]
}; # end proc

proc sendListOfShots_old_version {} {
    # Go to the facility directory and send a list of subdirectories
    # that contain the actual shot data.
    global td
    global contentString
    set dirName [file join $td(rootDir) $td(facility)]
    if { [file exists $dirName] } {
        cd $dirName
        foreach shotDir [glob *] {
            if { ![string equal $shotDir "rundesc"] && \
                ![string equal $shotDir "descript"] } {
                # send only real shot directory names
                append contentString "$shotDir\n"
            }; # end if
        }; # end foreach
    } else {
        append contentString \
            "td_server_error : Cannot find facility $facility directory.\n"
    }; # end if
}; # end proc

proc sendRunDescription {} {
    # Look for the Run Description text file in the
    # RUNDESC (or rundesc) directory.
    # Take care to look for both upper- and lower-case names.

    global td
    global contentString

    set dirPattern [join \
        [list $td(rootDir) / $td(facility) / {{rundesc,RUNDESC}}] "]"
    set dirList [glob -nocomplain $dirPattern]
    if { [llength $dirList] == 0 } {
        append contentString \
            "td_server_error : Could not find Run Descriptions directory.\n"
    }; # end if
    set dirName [lindex $dirList 0]

    # T4-style name
    set filePattern1 [join \
        [list $dirName / $td(shot) . {{txt,TEXT}}] "]"
    # X2-style name
    if { [string compare -nocase -length 1 $td(shot) s] == 0 } {
        # take off the first "s" or "S"
        set shortName [string range $td(shot) 1 end]
    } else {
        set shortName $td(shot)
    }; # end if
    set filePattern2 [join \
        [list $dirName / {{run,RUN}} $shortName . {{txt,TEXT}}] "]"
    set fileList [glob -nocomplain $filePattern1 $filePattern2]

```

```

if { [llength $fileList] == 0 } {
    append contentString \
        "td_server_error : No Run Description file for $td(shot).\n"
}; # end if
set fileName [lindex $fileList 0]

if { [file exists $fileName] } {
    set fp [open $fileName "r"]
    append contentString [read $fp]
    close $fp
} else {
    append contentString \
        "td_server_error : Run description file $fileName doesn't exist.\n"
}; # end if
}; # end proc

proc sendChannelList {} {
    global td
    global contentString

    if { [catch {exec gzip -d -c $td(dataFile) > $td(tempDataFile)} result] } {
        append contentString "td_server_error : gunzip command failed\n"
        append contentString "$result\n"
    } else {
        # we have successfully uncompressed the data file
        set fp [open $td(tempDataFile) "r"]
        append contentString [read $fp]
        close $fp
    }; # end if
}; # end proc

proc sendChannelData {} {
    global td
    global contentString
    # Open the file, read the metadata and
    # reconstruct the time values if they are not already there.

    # puts "send data from file $td(dataFile)"
    if { [catch {exec gzip -d -c $td(dataFile) > $td(tempDataFile)} result] } {
        append contentString "td_server_error : gunzip command failed\n"
        append contentString "$result\n"
    } else {
        # we have successfully uncompressed the data file
        set fp [open $td(tempDataFile) "r"]

        if { [string equal $td(part) raw] } {
            # Send the whole data file as it is.
            append contentString [read $fp]
        } else {
            # Process the data file a little more and
            # send only the requested piece.
            set commentLine [gets $fp]

            # Extract the header information and save it.
            while { [eof $fp] == 0 } {
                set headerLine [gets $fp]
                # discard comment character
                regsub {#} $headerLine {} headerLine
                # remove surplus whitespace
                set headerLine [string trim $headerLine]
                regsub -all {\s\s+} $headerLine {} headerLine
                if { [string length $headerLine] == 0 } {
                    # we have hit the blank line
                    break;
                } else {
                    foreach {name value} [split $headerLine] {

```

```

        set td(header.$name) $value
    }; # end for
}; # end if
}; # end while

if { [string equal $td(part) info] } {
    # Send back just the metadata/info
    foreach {item} [array names td "header.*" ] {
        regrab {header\.} $item {} itemLabel
        append contentString "$itemLabel [set td($item)]\n"
    }; # end foreach
} else {
    # Send back the data itself.
    set N $td(header.dataPoints)
    set dt $td(header.timeInterval)
    set t0 $td(header.timeStart)
    if {[string equal $td(header.withTimeColumn) "no"]} {
        set generateTime 1
    } else {
        set generateTime 0
    }; # end if
    for {set j 0} {$j < $N} {incr j} {
        set dataLine [gets $fp]
        if { [eof $fp] } break
        if { $generateTime } {
            set t [expr $j * $dt]
            set v [string trim $dataLine]
        } else {
            foreach {t v} [split $dataLine] {}
        }; # end if
        append contentString "$t $v\n"
    }; # end for
}; # end if
}; # end if

close $fp
}; # end if
}; # end proc

proc findShotDataDir {} {
    global td
    set dirName [file join $td(rootDir) $td(facility) $td(shot)]
    if { [file exists $dirName] } {
        set td(dataDir) $dirName
        return 1
    } else {
        set td(dataDir) ""
        return 0
    }; # end if
}; # end proc

proc findChannelDataFile {} {
    global td
    set fname $td(shot)
    if { [string equal $td(channel) list] } {
        append fname A.LST.gz
    } else {
        append fname A . $td(channel) .gz
    }; # end if
    set fileName [file join $td(dataDir) $fname]
    if { [file exists $fileName] } {
        set td(dataFile) $fileName
        return 1
    } else {
        set td(dataFile) ""
        return 0
    }
}; # end if
}; # end proc

```

```

    }; # end if
}; # end proc

proc processRequest {} {
    global td
    global contentString
    if { [string equal $td(shot) "list"] } {
        # We want a list of the shots for a particular facility.
        sendListOfShots
    } else {
        # We want the data for a particular shot.
        if { [findShotDataDir] } {
            if { [string equal $td(channel) "rundescription"] } {
                sendRunDescription
            } elseif { [findChannelDataFile] } {
                if { [string equal $td(channel) "list"] } {
                    sendChannelList
                } else {
                    sendChannelData
                }; # end if
            } else {
                append contentString "td_server_error : "
                append contentString "Channel $td(channel) not found.\n"
            }; # end if
        } else {
            append contentString "td_server_error : "
            append contentString "Facility: $td(facility) Shot: $td(shot) "
            append contentString "not found.\n"
        }; # end if
    }; # end if
}; # end proc

proc allowAccess {} {
    # Provides a basic access filter so that the generic guest
    # login that is embedded in the distributed client files
    # can look at a few example data but not more.
    # We can also use this procedure to apply an access policy
    # more generally.
    global td contentString allowedShotList
    if { $td(cgiScript) } {
        # Impose restrictions on the tdguest only.
        # All others let through by the web-server should be valid users.
        set user $::env(REMOTEUSER)
        if [info exists allowedShotList($user)] {
            set combinedName $td(facility)+$td(shot)
            set combinedList [set allowedShotList($user)]
            if { [lsearch -exact $combinedList $combinedName] >= 0 } {
                # restricted user but selection allowed
                return 1
            } else {
                # restricted user and invalid selection
                return 0
            }
        } else {
            # not a restricted user
            return 1
        }
    } else {
        # Don't impose any restrictions in a command-line environment.
        # The data must already be on the local disk.
        return 1
    }
}

#----- main script starts here -----

```

```
package require ncgi
package require base64

initializeData
parseQueryString
if { [allowAccess] } {
    processRequest
} else {
    append contentString "td_server_error : "
    append contentString "Access to that data is not allowed.\n"
}
sendContentString

# tidy up
catch { file delete $td(tempDataFile) }
```

# B Archive Maintenance Scripts

## B.1 td\_scan.tcl

```
#!/bin/sh
#\
exec tclsh "$0" ${1+"$@"}

# File:
# td_scan.tcl
#
# Purpose:
# Scan the shot directories and convert the binary data file
# that have been written by MONC into equivalent text files
# and write a shot.index file to aid the server script.
# We'll make use of the defrock program to do the actual conversion.
#
# Note that the original MONC files should remain intact.
#
# Usage:
# See below for procedure echoUsage.
#
# Author:
# Peter Jacobs
# Department of Mechanical Engineering
# The University of Queensland
#
# Revisions:
# 27-feb-02 original code
# 06-mar-02 added option to process a single shot,
# 13-mar-02 cope with X2 style MOD files
#
#----- preamble-----

if { [string length [info script]] } {
    set td(scriptHome) [file dirname [info script]]
} else {
    set td(scriptHome) [file dirname [info nameofexecutable]]
}; # end if
# puts "td(scriptHome) is $td(scriptHome)"
set td(logFileName) td_scan.log
# Locate the defrock directory assuming that we are starting
# in the tds directory.
set td(defrockDir) [file join [pwd] .. defrock]
# puts "td(defrockDir) is $td(defrockDir)"
if { [string equal $::tcl_platform(platform) windows] } {
    set td(defrockProgram) defrock
} else {
    set td(defrockProgram) ./defrock
}; # end if
# Remember where we started.
set td(workDir) [pwd]

#----- procedures-----

proc echoUsage {} {
    puts "Usage: td_scan.tcl dataDir action ?shot=shotId?"
    puts "where:"
    puts "dataDir is the name of the directory which contains"
    puts "         the individual shot directories."
    puts "action is one of new|refresh|erase|index"
    puts "         new option creates the ascii files if they are"
    puts "         not already present."
    puts "         refresh option forces the ASCII files to"
    puts "         be regenerated even if they are already present."
    puts "         erase option erases all of the ASCII files."
    puts "         index option writes an index file in dataDir"
    puts "By default, all shot directories are scanned."
```

```

    puts "A single shot can be scanned as specified with the"
    puts "shot=shotId argument."
    puts"-----"
}; # end proc

proc looksLikeAShotDir { shotDir } {
    # From the main data directory, look for the MONC header file
    # within the shot directory.
    # Try various upper- and lower-case combinations.
    set shotName1 $shotDir
    set pattern1 [join [list $shotDir / $shotName1 . {{hed,HED}}] "" ]
    set shotName2 [string toupper $shotDir]
    set pattern2 [join [list $shotDir / $shotName2 . {{hed,HED}}] "" ]
    set shotName3 [string tolower $shotDir]
    set pattern3 [join [list $shotDir / $shotName3 . {{hed,HED}}] "" ]
    set hedFileList [glob -nocomplain $pattern1 $pattern2 $pattern3]
    return [expr [llength $hedFileList] >= 1]
}; # end proc

proc fixFileNameCase { shotDir } {
    # From the main data directory, look to see if the MONC data
    # files have the same case as the directory name.
    # We assume that the shotName and the directory name are the same.
    set shotName $shotDir
    set saveDir [pwd]
    cd $shotDir
    set fileList [glob -nocomplain *]
    foreach f $fileList {
        set oldRootName [file rootname $f]
        set extn [file extension $f]
        if { ![string equal $oldRootName $shotName] } {
            # The names don't match exactly
            if { [string compare -nocase $oldRootName $shotName] == 0 } {
                # ...but they do match except for case of the letters
                catch { file rename $f [join [list $shotName $extn] ""] } junk
            } else {
                # This file name is not of the form shot.nnn; leave it.
            }; # end if
        }; # end if
    }; # end foreach
    cd $saveDir
}; # end proc

proc setDataFilePermissions { shotName } {
    # From the main data directory, look into the MONC data
    # directory for the shot and make sure that the data files
    # have reasonable permissions.
    # For some reason, defrock seems to need the files to be
    # writeable as well as readable.
    set fileList [glob -nocomplain $shotName/*]
    foreach f $fileList {
        # r=1 w=1 x=0 --> 6
        file attributes $f -permissions 0664
    }; # end foreach
}; # end proc

proc checkForModFile { shotName } {
    # Assume that we are starting from the main data directory.
    # Look for one of the forms of the MOD file.
    # Returns a value of 1 if a standard MOD file already exists
    # or can be copied from a similarly-named MOD file.
    cd descript
    set foundModFile 0
    # Initially, look for a standard (T4) style MOD file name.

```

```

set pattern1 [join [list $shotName . {{mod,MOD}}] "]
set fileList [glob -nocomplain $pattern1]
if { [llength $fileList] == 0 } {
    # Didn't find a T4 standard MOD file.
    # Allow for upper- and lower-case variations.
    set pattern2 [join [list [string toupper $shotName] . {{mod,MOD}}] "]
    set pattern3 [join [list [string tolower $shotName] . {{mod,MOD}}] "]

    # Secondly, look for an X2 standard
    # MOD file that has the shot Id starting with "s" or "S"
    # while the MOD filename drops this first character.
    if { [string compare -nocase -length 1 $shotName s] == 0 } {
        # Try removing that first "s".
        set shotName2 [string range $shotName 1 end]
        set pattern1 [join [list $shotName2 . {{mod,MOD}}] "]
    } else {
        # Try adding an "s".
        set pattern1 [join [list {[sS]} $shotName . {{mod,MOD}}] "]
    }; # end if

    set fileList [glob -nocomplain $pattern1 $pattern2 $pattern3]
    if { [llength $fileList] >= 1 } {
        # Found a MOD file with a nonstandard name. Copy it.
        set sourceFile [lindex $fileList 0]
        set destinationFile [join [list $shotName .MOD] "]
        file copy $sourceFile $destinationFile
        set foundModFile 1
    } else {
        # Didn't find an X2 style MOD file, cant' do much else.
        set foundModFile 0
    }; # end if
} else {
    set foundModFile 1
}; # end if
cd ..
return $foundModFile
}; # end proc

```

```

proc ASCIIFilesArePresent { shotName } {
    # From the main data directory, look into the shot directory
    # to see if the ASCII files are already present in compressed form.
    # Test 1: look for the LST file
    # Test 2: there should be at least one other compressed data file.
    # Both tests need to be true to return a true result.
    set asciiName [join [list $shotName A] "]
    set i1 [file exists $shotName/$asciiName.LST.gz]
    set fileList [glob -nocomplain $shotName/$asciiName*gz]
    set i2 [expr [llength $fileList] >= 2]
    return [expr $i1 && $i2]
}; # end proc

```

#-----start main script-----

```

# Command-line arguments
if { $argc < 2 } {
    echoUsage
    exit
};

set td(dataDir) [lindex $argv 0]
if { ![file isdirectory $td(dataDir)] } {
    puts "$td(dataDir) is not a directory."
    exit
}; # end if

set td(commandOption) [lindex $argv 1]

```

```

if { [lsearch {new refresh erase index} $td(commandOption)] == -1 } {
    puts "Invalid action : $td(commandOption)"
    exit
}; # end if

if { $argc >= 3 } {
    # Assume that the third command line argument
    # is there to specify one shot.
    set td(specificShotId) [lindex [split [lindex $argv 2] "="] 1]
} else {
    set td(specificShotId) ""
}; # end if

set logFile [open $td(logFileName) w]
puts $logFile "dataDir is $td(dataDir)"
if { [string equal $td(commandOption) erase] } {
    puts $logFile "Will erase all ASCII files from shot directories."
} elseif { [string equal $td(commandOption) refresh] } {
    puts $logFile "Will refresh ASCII files in shot directories."
} elseif { [string equal $td(commandOption) new] } {
    puts $logFile "Will generate ASCII files in shot directories "
    puts $logFile "only where they don't yet exist."
} elseif { [string equal $td(commandOption) index] } {
    puts $logFile "Will generate a shot.index file."
} else {
    puts "Should not have reached this point."
    puts "td(commandOption) is $td(commandOption)"
    close $logFile
    exit
}; # end if

# Now, sift through the directories within td(dataDir) and
# process those subdirectories that contain shot data.

cd $td(dataDir)
if { $td(specificShotId) == "" } {
    set directoryList [glob -nocomplain *]
} else {
    set directoryList $td(specificShotId)
}; # end if
# puts "directoryList: $directoryList"

# filter out directories that don't contain shot data
foreach dirName $directoryList {
    if { ![looksLikeAShotDir $dirName] } {
        set i [lsearch $directoryList $dirName]
        if { $i >= 0 } {
            set directoryList [lreplace $directoryList $i $i]
        }; # end if
    }; # end if
}; # end foreach

if { [string equal $td(commandOption) index] } {
    # Assming that we are already in $td(dataDir),
    # write the shot.index file with one shot id per line.
    set indexFile [open "shot.index" w]
    puts $indexFile [join [lsort $directoryList] "\n"]
    close $indexFile
    puts $logFile "Index file written."
    close $logFile
    exit
}; # end if

puts $logFile "----- Start of Report -----"
foreach shotName $directoryList {
    puts -nonewline $logFile "shot $shotName "
    puts -nonewline " $shotName "
}

```

```

flush stdout

# Assume that the files do not already exist and
# that we want to make them.
set doEraseASCIIFiles 0
set doMakeASCIIFiles 1

if { [string equal $std(commandOption) erase] } {
    set doEraseASCIIFiles 1
    set doMakeASCIIFiles 0
}; # end if

if { [string equal $std(commandOption) new] && \
    [ASCIIFilesArePresent $shotName] } {
    puts -nonewline $logFile \
        "; ASCII files are already present; do nothing"
    set doEraseASCIIFiles 0
    set doMakeASCIIFiles 0
}; # end if

if { [string equal $std(commandOption) refresh] } {
    set doEraseASCIIFiles 1
    set doMakeASCIIFiles 1
}; # end if

if { $doEraseASCIIFiles } {
    puts -nonewline $logFile "; erase ASCII files"
    cd $shotName
    set targetPattern [join [list $shotName A * gz] ""]
    foreach targetFile [glob -nocomplain $targetPattern] {
        # puts "erase file: $targetFile"
        file attributes $targetFile -permissions 0666
        file delete -force $targetFile
    }; # end foreach
    cd ..
}; # end if

if { $doMakeASCIIFiles } {
    if { [checkForModFile $shotName] } {
        # Fix up the file names when shot directory name
        # doesn't have the same case as the files within it.
        fixFileNameCase $shotName

        # Make sure that we have appropriate permissions on the
        # MONC data files.
        # For some reason, defrock needs them to be writeable.
        if { [string equal $::tcl_platform(platform) windows] } {
            # do nothing on windows
        } else {
            setDataFilePermissions $shotName
        }; # end if

        # Build the ASCII files from the MONC files.
        # Be careful with where we start running defrock.
        # It needs the monc.val file to be in the same
        # directory as it is started.
        set saveDir [pwd]
        cd $std(defrockDir)
        catch { exec $std(defrockProgram) $std(dataDir) $shotName } \
            defrockOutput
        # puts $defrockOutput
        cd $saveDir

        if { [regexp "End of defrock run." $defrockOutput] } {
            puts -nonewline $logFile "; defrock successful"
        } else {
            puts -nonewline $logFile "; defrock unsuccessful"
        }; # end if
    }
}; # end if

```

```

    puts -nonewline $logFile "; gzip ASCII files (if any)"
    cd $shotName
    set targetPattern [join [list $shotName A *] ""]
    foreach targetFile [glob -nocomplain $targetPattern] {
        catch { exec gzip -f $targetFile } gzipOutput
        # puts "; file $targetFile, gzipOutput is $gzipOutput"
    }; # end for
    # Tidy up, just in case we have dumped core here.
    file delete -force core
    cd $saveDir
} else {
    puts -nonewline $logFile "; cannot find MOD file."
}; # end if
}; # end if

    puts $logFile "." ; # to terminate the line
}; # end foreach

puts "End of scan."
exit
#-----end main script-----

```

## B.2 update\_monc\_files.tcl

```

#!/bin/sh
# update_monc_files.sh
#
# Looks in the (remote) UQSPACE directories and updates
# the local copy of the MONC data files for each facility.
# Once the files have been updated and tidied up,
# it runs the td_scan.tcl script.
#
# PJ, 28-Apr-03
#

SRC=/mnt/uqspace
DEST=/home2/moncdata
FACILITY_LIST="T4 X2 X3"

#-----
echo "Stage 1: copy new files from UQSPACE backup area."

if ! [ -d $SRC/t4/data ]
then
    echo "$SRC directories cannot be seen."
    echo "Use ncpmount to make them accessible."
    exit -1
else
    echo "$SRC directories are visible, let us do some work."
fi

if ! [ -d $DEST ]
then
    echo "Cannot see the destination directory $DEST."
    exit -1
else
    echo "Can also see the destination directory $DEST."
fi

echo "T4:"
rsync -av $SRC/t4/data/* $DEST/T4/
echo "X3:"
rsync -av $SRC/xtube/X3/* $DEST/X3/
echo "X2:"
rsync -av $SRC/xtube/X2/* $DEST/X2/

#-----

```

```

echo "Stage 2: Clean-up."

CRAP_LIST="T4/5848/5849"
for CRAP in $CRAP_LIST
do
    echo "    Checking for $DEST/$CRAP"
    if [ -d $DEST/$CRAP ]
    then
        echo "    Deleting $DEST/$CRAP"
        rm -rf $DEST/$CRAP
    fi
done

echo "Copy some of the X3 files to lower-case directories."
rsync -av $DEST/X3/RUNDESC/* $DEST/X3/rundesc/
rsync -av $DEST/X3/DESCRIPT/* $DEST/X3/descript/

#-----
echo "Stage 3: scanning for new MONC files in the archive."

if [ -f td_scan.tcl ]
then
    TODAYS_DATE="date +%d%b%y"
    echo "Todays Date = $TODAYS_DATE"
    for FACILITY in $FACILITY_LIST
    do
        echo "    Scanning for Facility $FACILITY"
        ./td_scan.tcl $DEST/$FACILITY/ new
        mv td_scan.log $DEST/$FACILITY/td_scan_$TODAYS_DATE.log
        ./td_scan.tcl $DEST/$FACILITY/ index
    done

    for FACILITY in $FACILITY_LIST
    do
        echo "Sending $FACILITY to web server; Enter password at the prompt."
        rsync -av -e ssh $DEST/$FACILITY proba:/home2/moncdata/
    done
else
    echo "Cannot see the td_scan.tcl script in the current directory."
    exit -1
fi

```

## C MATLAB/Octave Client Scripts

Machine-readable copies of these files are available online [4]. Although both sets of functions (external process and Java-based access) work in recent versions of MATLAB, only the external process functions work in Octave. There are also slight differences in the containers used for the return of the metadata name-value pairs. In Octave, lists are used but, in MATLAB, cell-arrays are used. The core of the function is the assembly of a command line, the calling of the external program/process to talk to the web server and the collation of the data. Much of the code bloat is caused by error checking and by subtle differences between Octave and MATLAB. If you intend to operate in only one environment and don't care for the error checking, a much simpler function is possible.

### C.1 Web Access via an External Process

#### C.1.1 get\_data.m

```
function [t, v] = get_data( facilityName, shotName, channelName, partName )
% GET_DATA gets specified data from the Tunnel-Data Server.
% Input:
% facilityName : string, e.g. 'T3', 'T4', 'X1', 'X2' or 'X3'
% shotName     : string specifying the base-file-name for the shot.
% channelName  : string specifying the channel number.
% partName     : string, either 'data' or 'info', specifying
%               which part of the data file that we want to get.
% Output:
% Depending on what is requested, the returned value may be:
% (1) the data as two numeric-arrays (vectors)
% (2) the metadata as two cell-arrays (or lists in Octave) of strings
% (3) the shot list as a cell-array (or lists in Octave) of strings
% (4) the channel names and Ids as two cell-arrays (or lists in Octave) of strings.

% PJ, 26-dec-01, 22-Apr-03

% First, decide what our action is to be for this call.
if strcmp( shotName, 'list' ) == 1
    action = 'getShotList';
    channelName = 'dummy';
    partName = 'dummy';
elseif strcmp( channelName, 'list' ) == 1
    action = 'getChannelList';
    partName = 'dummy';
elseif strcmp( partName, 'info' ) == 1
    action = 'getChannelInfo';
else
    action = 'getChannelData';
end

% select the mode of interaction with the HTTP/CGI server
% 0 = wget
% 1 = tcl_web_client
useTclClientScript = 0;
% Matlab won't know about the predefined constant OCTAVE_VERSION
inOctave = exist( 'OCTAVE_VERSION' ) > 0;

% Second, put together the command line options and then
% run an external process to get the data from the server.

if useTclClientScript == 1
```

```

% If wget is not available, there is a TCL client script
% that can talk to the Tunnel-Data Server via HTTP.
cmd = ['./td_web_client.tcl_', facilityName, '_-', shotName, '_-', ...
       channelName, partName]
else
% Set up a URL for wget to do the work.
server = 'www.mech.uq.edu.au';
% server = 'galaxy5'; % test server
userName = 'tdguest';
password = 'tdpasswd';
cgiCall = ['/cgi-bin/tds/td_server.tcl?facility=', facilityName, ...
          '+shot=', shotName, ...
          '+channel=', channelName, ...
          '+part=', partName];
cmd = ['wget', '_--output-document=temporary.mat', ...
      '_--http-user=', userName, '_--http-passwd=', password, ...
      '_--quiet', ...
      '_http://', server, cgiCall];
end
% disp( cmd );
if inOctave
% Use an Octave style system call.
[result, status] = system(cmd);
else
% Matlab style
[status, result] = system(cmd);
end
if status ~= 0
disp( 'The_external_process_did_not_run_successfully.' );
disp( 'Maybe_the_server_could_not_be_found.' );
t = [];
v = [];
return;
end

% Before trying to pick up the requested data,
% peek at the first line of the temporary file
% to see if the Tunnel-Data Server sent an error message.
[fid, msg] = fopen( 'temporary.mat', 'r' );
line_of_text = fgetl( fid );
fclose( fid );
if ~isempty( findstr( line_of_text, 'td_server_error' ))
disp( 'The_data_that_was_requested_could_not_be_found.' );
disp( 'The_server_returned_the_following_message:' );
disp( line_of_text );
t = [];
v = [];
return;
end

% Finally, pick up the local file containing the returned data
% and separate it into its components so that it can be returned.
if strcmp( action, 'getChannelData' ) == 1
% The temporary file contains two columns of numbers.
% Split into two numeric arrays (vectors) for return.
if inOctave
load temporary.mat;
else
load -ascii temporary.mat;
end
t = temporary(:,1);
v = temporary(:,2);
elseif strcmp( action, 'getChannelInfo' ) == 1 || ...
strcmp( action, 'getChannelList' ) == 1
% The temporary file contains lines with two words each.
count = 0;
if inOctave
t = list; v = list; % use lists

```

```

else
    t = {}; v = {}; % use cell-arrays
end
[fid, msg] = fopen( 'temporary.mat', 'r' );
line_of_text = fgetl( fid );
while ~isnumeric( line_of_text )
    count = count + 1;
    if inOctave
        [a, b, n] = sscanf( line_of_text, '%s_%s', 'C' );
        t = append( t, a );
        v = append( v, b );
    else
        [a, b] = strread( line_of_text, '%s_%s' );
        t{count} = char(a);
        v{count} = char(b);
    end
    line_of_text = fgetl( fid );
end
fclose( fid );
elseif strcmp( action, 'getShotList' ) == 1
% The temporary file contains a collection of shot names,
% 10 per line, I believe.
count = 0;
if inOctave
    t = list; v = list; % use lists
else
    t = {}; v = {}; % use cell-arrays
end
[fid, msg] = fopen( 'temporary.mat', 'r' );
line_of_text = fgetl( fid );
while ~isnumeric( line_of_text )
    count = count + 1;
    if inOctave
        t = append( t, line_of_text );
    else
        t{count} = line_of_text;
    end
    line_of_text = fgetl( fid );
end
fclose( fid );
end
end

```

### C.1.2 td\_web\_client.tcl

```

#!/bin/sh
#\
exec tclsh "$0" "${1+"$@"}

# File:
# td_web_client.tcl
#
# Purpose:
# This script (when called from octave, say) will go and
# ask for some data from the td_server.tcl CGI script.
#
# Usage:
# td_web_client.tcl <facility> <shot> <channel> <part>
#
# Author:
# Peter Jacobs
# Department of Mechanical Engineering
# The University of Queensland
#
# Revisions:
# 26-Dec-01 original code
# 29-Dec-01 extra command-line options
# 30-Dec-01 add documentation

package require http

```

```

# the facility name and shot number should have been passed to
# this script as command line arguments
if { $argc == 0 } {
    # no arguments passed, set defaults
    set facility T4
    set shot 7319
    set channel 622
    set part data
} else {
    # octave seems to pass only one argument on the command line.
    # It may have spaces and so can be pulled apart as a list.
    set facility [lindex $argv 0]
    set shot [lindex $argv 1]
    set channel [lindex $argv 2]
    set part [lindex $argv 3]
}; # end if

set queryString "facility=$facility"
append queryString "+shot=$shot"
append queryString "+channel=$channel"
append queryString "+part=$part"

set serverIP "130.102.240.102"
set serverURL "http://$serverIP"
append serverURL "/cgi-bin/td_server.tcl?"
append serverURL $queryString

set matFile temporary.mat

set r [http::geturl $serverURL]
set rcode [lindex [http::code $r] 1]

if { $rcode == 200 } {
    # we received the data OK
    set rdata [http::data $r]
    set fp [open $matFile w]
    puts $fp $rdata
    close $fp
} else {
    puts "Data transfer failed with http code $rcode"
}; # end if

```

### C.1.3 test\_data\_server.m

```

% test_data_server.m
% Show how the data can be fetched from the Tunnel Data Server.
facility = 'T4'; shot_id = '7319'; channel = '110';
useJava = 0;

% This script can be used in both MATLAB and Octave.
% Because there are differences, the script needs to be aware
% of its environment.
inOctave = exist( 'OCTAVE_VERSION' ) > 0;
if inOctave
    % The Java VM is only accessible from MATLAB 6.x.
    useJava = 0;
end

% Here is the actual interaction with the Tunnel-Data Server.
tic;
if useJava
    disp( 'Talk to Tunnel-Data-Server via Java-VM.' );
    disp( 'First, get the metadata.' );
    [attrib, value] = fetch_channel_header( facility, shot_id, channel );
    disp( 'Now, get the actual data.' );
    [t, v] = fetch_channel_data( facility, shot_id, channel );
else
    disp( 'Talk to the Tunnel-Data-Server via a system call.' );

```

```

disp( 'First, get the metadata.' );
[attrib, value] = get_data( facility, shot_id, channel, 'info' );
disp( 'Now, get the actual data.' );
[t, v] = get_data( facility, shot_id, channel, 'data' );
end
disp( sprintf( 'Elapsed time %f seconds.', toc ) );

% Build the plot titles and axis labels.
% Make use of the known order in which the metadata is returned.
subplot( 2, 1, 1 );
if inOctave
    % metadata is in lists
    my_title = [ 'Facility ', facility, ', Shot ', nth(value,9), ...
                ', Channel ', nth(value,14) ];
    my_xlabel = [ 'time in ', nth(value,11) ];
    my_ylabel = [ nth(value,5), ' in ', nth(value,7) ];
    % set up annotation and then plot
    axis( [6000, 16000, 0, 40000] );
    title( [my_title, ' Raw Data' ] );
    xlabel( my_xlabel ); ylabel( my_ylabel );
    plot( t, v );
else
    % For MATLAB, metadata is in cell-arrays
    my_title = [ 'Facility ', facility, ', Shot ', char(value{9}), ...
                ', Channel ', char(value{14}) ];
    my_xlabel = [ 'time in ', char(value{11}) ];
    my_ylabel = [ char(value{5}), ' in ', char(value{7}) ];
    % first plot then annotate.
    plot( t, v );
    axis( [6000, 16000, 0, 40000] );
    title( [my_title, ' Raw Data' ] );
    xlabel( my_xlabel ); ylabel( my_ylabel );
end

disp( 'Filter and plot data again.' );
try
    % Attempt to use the signal-processing toolbox function.
    [b, a] = butter( 2, 0.05 );
catch
    % But, if it is not available, set the filter coefficients manually.
    b = [0.0055427 0.0110854 0.0055427];
    a = [1.0 -1.7786318 0.8008026];
end
vf = filter( b, a, v );

subplot( 2, 1, 2 );
if inOctave
    % set up annotation and then plot
    axis( [6000, 16000, 0, 40000] );
    title( [my_title, ' Filtered Data' ] );
    xlabel( my_xlabel ); ylabel( my_ylabel );
    plot( t, vf );
else
    % for MATLAB plot then annotate.
    plot( t, vf );
    axis( [6000, 16000, 0, 40000] );
    title( [my_title, ' Filtered Data' ] );
    xlabel( my_xlabel ); ylabel( my_ylabel );
end

disp( 'Averages during test-time.' );
tt = t > 7500 & t <= 8000;
m = sum( v .* tt ) / sum( tt );
mf = sum( vf .* tt ) / sum( tt );
disp( sprintf( 'Raw: %f, Filtered: %f', m, mf ) );

```

## C.2 Java-based Network Access

### C.2.1 fetch\_text\_from\_server.m

```
function content_text = fetch_text_from_server( ...
    facilityName, shotName, channelName, partName )
% FETCHTEXTFROMSERVER fetches the requested data from the server.
% Input:
% facilityName : string, e.g. 'T3', 'T4', 'X1', 'X2' or 'X3'
% shotName     : string specifying the base-file-name for the shot.
% channelName  : string specifying the channel number.
% partName     : string, either 'data' or 'info', specifying
%               which part of the data file that we want to get.
% Output:
% Returns the data as a cell-arrays of strings.

% Peter J. 19-April-03

protocol = 'http';
% Set the following to appropriate values.
% host_name = '192.168.0.102';
host_name = 'www.mech.uq.edu.au';
with_passwd = 1;
user_name = 'tdguest';
passwd = 'tdpasswd';

% Set up the combined script-name and query string from the supplied pieces.
% It is not necessary to send shot and part names.
if isempty(partName) && isempty(channelName)
    cgi_script_name = sprintf( ...
        '/cgi-bin/tds/td_server.tcl?facility=%s+shot=%s', ...
        facilityName, shotName );
elseif isempty(partName)
    cgi_script_name = sprintf( ...
        '/cgi-bin/tds/td_server.tcl?facility=%s+shot=%s+channel=%s', ...
        facilityName, shotName, channelName );
else
    cgi_script_name = sprintf( ...
        '/cgi-bin/tds/td_server.tcl?facility=%s+shot=%s+channel=%s+part=%s', ...
        facilityName, shotName, channelName, partName );
end

url = java.net.URL( protocol, host_name, cgi_script_name );
if with_passwd == 1
    % Encode the username and passwd before setting the Authorization
    connection = url.openConnection();
    text_string = [ user_name, ':', passwd ];
    % encoded_string = base64encode( text_string ); % not working
    encoded_string = 'dGRndWVzdDp0ZHBhc3N3ZA=='; % encoded manually
    connection.setRequestProperty( 'Authorization', [ 'Basic ', encoded_string ] );
    connection.connect();
    is = connection.getInputStream();
else
    % We will operate without setting the Authorization property.
    is = openStream( url );
end

isr = java.io.InputStreamReader( is );
ibr = java.io.BufferedReader( isr );

line_of_text = char( readLine( ibr ) );
count = 0;
content_text = {}; % Use a cell-array to collect the contents
while ~isempty(line_of_text)
    count = count + 1;
    content_text{count} = line_of_text;
    line_of_text = char( readLine( ibr ) );
end
```

## C.2.2 fetch\_channel\_data.m

```
function [t, v] = fetch_channel_data( facilityName, shotName, channelName )
% FETCH_CHANNEL_DATA fetches the time-value pairs.
% Input:
% facilityName : string, e.g. 'T3', 'T4', 'X1', 'X2' or 'X3'
% shotName     : string specifying the base-file-name for the shot.
% channelName  : string specifying the channel number.
% Output:
% Returns the data as two numeric arrays (vectors).

% Peter J. 19-April-03

% Collect the data text as a cell array of strings,
% one per line of the original header.
data_text = fetch_text_from_server( facilityName, shotName, channelName, 'data' );

% Split the strings into time and value components.
count = 0; t = zeros(20000,1); v = zeros(20000,1);
for line_text = data_text
    count = count + 1;
    [time_stamp value] = strread( char(line_text), '%s_%s' );
    % At this point, it appears that time_stamp and
    % value are both cell variables.
    t(count) = sscanf( char(time_stamp), '%f' );
    v(count) = sscanf( char(value), '%f' );
end

% Trim the arrays to the true size of the data.
t = t(1:count);
v = v(1:count);
```

## C.2.3 fetch\_channel\_header.m

```
function [names, values] = fetch_channel_header( facilityName, shotName, channelName )
% FETCH_CHANNEL_DATA fetches the channel metadata.
% Input:
% facilityName : string, e.g. 'T3', 'T4', 'X1', 'X2' or 'X3'
% shotName     : string specifying the base-file-name for the shot.
% channelName  : string specifying the channel number.
% Output:
% Returns the name-value pairs as two cell-arrays of strings.

% Peter J. 19-April-03

% Collect the header text as a cell array of strings,
% one per line of the original header.
header_text = fetch_text_from_server( facilityName, shotName, channelName, 'info' );

% Split the strings into name and value components.
count = 0; names = {}; values = {};
for line_text = header_text
    count = count + 1;
    [a, b] = strread( char(line_text), '%s_%s' );
    names{count} = char(a);
    values{count} = char(b);
end
```

## C.2.4 demonstrate\_fetch.m

```
% demonstrate_fetch.m
% Show how the data can be fetched from the Tunnel Data Server.

facility = 'T4'; shot_id = '7319'; channel = '110';

disp( 'First, get the metadata for the channel.' );
% and make use of the known order in which it is returned.
[attrib, value] = fetch_channel_header( facility, shot_id, channel );
my_title = ['Facility_', facility, 'Shot_', char(value{9}), ...
            'Channel_', char(value{14})];
```

```

my_xlabel = ['time_in_', char(value{11})];
my_ylabel = [char(value{5}), '_in_', char(value{7})];

disp( 'Now, get the actual data and display graphically.' );
[t, v] = fetch_channel_data( facility, shot_id, channel );
subplot( 2, 1, 1 ); plot( t, v );
axis( [6000, 16000, 0, 40000] );
title( my_title ); xlabel( my_xlabel ); ylabel( my_ylabel );

disp( 'Filter and display again.' );
try
    % Attempt to use the signal-processing toolbox function.
    [b, a] = butter( 2, 0.05 );
catch
    % Set the filter coefficients manually.
    b = [0.0055427 0.0110854 0.0055427];
    a = [1.0 -1.7786318 0.8008026];
end
vf = filter( b, a, v );
subplot( 2, 1, 2 ); plot( t, vf );
axis( [6000, 16000, 0, 40000] );
xlabel( my_xlabel ); ylabel( my_ylabel );

disp( 'Averages during test-time.' );
tt = t > 7500 & t <= 8000;
m = sum( v .* tt ) / sum( tt );
mf = sum( vf .* tt ) / sum( tt );
disp( sprintf( 'Raw:_%f, Filtered:_%f', m, mf ) );

```