Typed Data Transfer (TDT) User's Guide

Ciaron Linstead <linstead@pik-potsdam.de>

10th August 2004

Revision History:

04/03/2002 CL - version 0.1

12/09/2003 CL - version 0.2, extensive revision

29/07/2004 CL - version 0.3, converted master document to LaTeX from LyX

1 Introduction

The Typed Data Transfer (TDT) Library provides a simple, consistent interface for the transmission of data between programs in a platform- and language-independent way. It moves the complexities of handling data types and data sources into a self-contained library of functions.

In this way, complex data types (i.e. data types composed of elements with different data types, like a "struct" in C) can be passed between TDT-enabled programs with a single function call.

The TDT library also takes care of byte swapping when transferring data between big-endian and little-endian architectures.

The speed of transferring blocks of homogenous data (like arrays) is practically the same as with the non-TDT method.

The flexibility of the TDT approach means that modules can be coupled in various configurations and by various communications methods simply by altering configuration files: no re-compilation of modules is necessary.

The TDT functions are written in C, and are provided with Fortran interface functions for using the library in Fortran programs. Opening and closing of sockets and files are handled by TDT functions, and data is written or read by means of a call to the appropriate TDT function.

Apart from adding function calls the code, a programmer must also provide an XML (eXtensible Markup Language) description of the data to be transferred

and a configuration file for each program, also in XML. Each data structure being transferred needs its own XML description, each of which may be in separate XML files, or in just one.

The purpose of this document is to explain the use of the TDT function library when writing programs in C or Fortran. Some example programs are included and will be explained.

2 Using the TDT in C

2.1 Prerequisites

2.1.1 "#include"s

To include the TDT library, simply

```
#include "tdt.h"
```

2.1.2 Declarations

It is necessary to declare a variable of type `TDTState`. This variable serves as a unique identifier for each data item being transferred, by bundling an XML description and a descriptor for the communication channel.

A variable of type `TDTConfig` is also required. This is used to refer to the data which will be read from a configuration file.

Examples:

```
TDTState ts;
TDTConfig tc;
```

2.2 User functions

2.2.1 tdt_configure()

Purpose

Read and parse the specified configuration file.

Input parameter

```
String config_filename /* the name of the config file*/
                       /* to be parsed */
```

Output parameter

```
TDTConfig tc /* a parsed copy of the configuration file */
```

Example:

```
tc = tdt_configure("config.xml");
```

2.2.2 tdt_open()

Purpose

`tdt_open()` opens the communication channel specified by the connection name

Input parameters

```
TDTState ts             /* the TDTState variable  */

TDTConfig tc            /* the TDTConfig variable */

String connection_name  /* the connection to open */
```

Output parameter

```
TDTState ts             /* a completed TDTState variable */
```

Example:

```
ts = tdt_open (ts, tc, "client_to_server");
```

2.2.3 tdt_read()

Purpose

To read the data specified by the given XML identifier, from the connection given in the `TDTState` parameter.

Input parameters

```
TDTState ts         /* identifier for this data transfer */

void *value         /* where to store the incoming data */

String name         /* the name of the data to be read, */
                    /* as it appears in the datadesc */
```

Output parameters

None.

Example

```
tdt_read (ts, &astruct, "astruct");
```

2.2.4 tdt_write()

Purpose

To write data to the connection given by the `TDTState` parameter as per the XML identifer string (parameter `"name"`).

Input parameters

```
TDTState ts          /* identifier for this data transfer */

void *value          /* pointer to the data being written */

String name          /* the name of the data to be written, */
                     /* as it appears in the datadesc */
```

Output parameters

None.

Example

```
tdt_write (ts, &astruct, "astruct");
```

2.2.5 tdt_size_array()

Purpose

This function is used to redefine the size of an array given in a datadesc. If your program uses dynamically sized arrays, you probably won't know in advance what size the array is when you create the datadesc XML. In this case, you can set the array size to zero in the datadesc and adjust the size later.

Input parameters

```
TDTState ts         /* identifier for this data transfer */

String name         /* name of the array being resized */

int size            /* the new size of the array */
```

Output parameters

None.

Example

```
tdt_size_array (ts, "dyn", new_int);
```

2.2.6 tdt_close()

Purpose

Closes the connection or open files and frees TDT-allocated memory.

Input parameters

```
TDTState ts             /* The connection to be closed */
```

Output parameters

None.

Example

```
tdt_close (ts);
```

2.2.7 tdt_end()

Purpose

Frees memory allocated by the TDT for the configuration information.

Input parameters

```
TDTConfig tc            /* The config data to be freed */
```

Output parameters

None.

Example

```
tdt_end (tc);
```

2.3 Compiling and linking

2.3.1 Compiling the library

The TDT can be compiled as a static library, `libtdt.a`. This must be in the library path of the development environment, or a directory specified by the `-L` flag in gcc. A makefile is included with the source files. Running `"make lib"` from within the `tdt` source directory will rebuild the TDT library.

2.3.2 Linking with your own programs

First build the library `libtdt.a` as explained above.

Compile your application...

```
gcc -c -I<location of tdt.h> -Wall <program>.c
```

... and link:

```
gcc <program>.o -L<location of TDT library> \
                -ltdt -lexpat_linux -o<program>
```

where `<program>.o` is the output from the compilation of `<program>.c`

This assumes that the Expat XML parser library is available system-wide on your machine. If it is not, and you want to use the Expat library supplied with the TDT, use the following link command:

```
gcc <program>.o -L<location of TDT library> -ltdt \
    -L<location of Expat library> -lexpat -o<program>
```

2.4 An example program

The following client program and it's associated server and a Makefile can be found in the `tdt/tests` subdirectory of the TDT distribution.

```
1  /* CLIENT (WRITER) EXAMPLE */
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include "tdt.h"
6
7  int
8  main (int argc, char **argv) {
```

```
9
10      /* a loop counter */
11      int i;
12
13      /* declare the variables we're going to write */
14      double vardouble;
15
16      struct {
17          int    elem1[2];
18          double elem2;
19      } astruct;
20
21      int varint;
22      double *dynarray;
23
24      /* declare some variables required by TDT */
25      TDTConfig tc;
26      TDTState  ts;
27
28      /* Now make up some meaningful data to write... */
29      astruct.elem1[0] = 10;
30      astruct.elem1[1] = 20;
31      astruct.elem2 = 199.99;
32
33      vardouble = 123.45;
34      varint = 12345;
35
36      dynarray = (double *) malloc (varint * sizeof (double));
37      for (i = 0; i < varint; i++) {
38          dynarray[i] = i * 0.33;
39      }
40
41      /* Get the configuration */
42      tc = tdt_configure ("clntconf.xml");
43
44      /* Establish the connection */
45      ts = tdt_open (ts, tc, "clnt_to_serv");
46
47      /* now write the data */
48      tdt_write (ts, &vardouble, "vardouble");
49      tdt_write (ts, &astruct, "astruct");
50      tdt_write (ts, &varint, "varint");
51
52      /* we haven't yet told TDT how big dynarray will be, so do that now ... */
53      tdt_size_array (ts, "dyn", varint);
54
```

```
55      /* ... and write it */
56      tdt_write (ts, dynarray, "dyn");
57
58      /* tidy up, close connections etc. */
59      tdt_close (ts);
60      tdt_end (tc);
61
62      return 0;
63  }
```

The TDT-specific lines to note are these:

5: Includes `tdt.h`

25, 26 : Declare `TDTState` (for connection-specific configuration) and `TDTConfig` (for program-specific configuration)

42: Calling `tdt_configure()` gets all the configuration data required

45: Open the connection. `clnt_to_serv` is the name in `clntconf.xml` which uniquely identifies the connection to open.

48, 49, 50: Each call to `tdt_write()` does just that: it write the data. On line 48 for example, the variable `vardouble` is written to connection `ts`, and the description comes from `vardouble`. Note that the name of the decl in the data description does not have to be the same as that of the variable, but it makes it clearer if it is.

53: Using `tdt_size_array` to dynamically size an array. Note that we've already written the variable `varint` to the server. It will use this variable in a similar call to `tdt_size_array` before reading the dynamic array.

59, 60: Tidying up. `tdt_close` closes the connection `ts`. We could re-open it later, but not after we've called `tdt_end` which removes the configuration information from memory.

2.5 Error conditions

The user functions of the TDT will crash, returning with error code 1, if an error occurs. It will also output a debugging message to `stderr`.

3 Using the TDT in Fortran

3.1 Prerequisites

3.1.1 Declarations

The following declarations are needed:

```
INTEGER tdtstate
INTEGER tdtconfig
```

3.2 User functions

3.2.1 tdt_fconfigure()

Purpose

Read and parse the specified configuration file.

Input parameter

```
C the name of the configuration file
configfilename
```

Output parameter

```
C a parsed copy of the configuration information
tdtconf
```

Example

```
CALL tdt_fconfigure(tdtconf, "config.xml");
```

3.2.2 tdt_fopen()

Purpose

`tdt_fopen()` opens the communication channel specified by the connection name.

Input parameters

```
C a TDTState variable
tdtstate

C a TDTConfig variable
tdtconf

C the connection name
connection
```

Output parameter

```
C a completed TDTState variable
tdtstate
```

Examples:

```
CALL tdt_fopen (tdtstate, tdtconf, "client_to_server")
```

3.2.3 tdt_fwrite()

Purpose

To write data to the connection given by the `TDTState` parameter as per the XML identifer string (parameter `"name"`).

Input parameters

```
C The identifier for this data transfer
tdtstate

C the data being written
val

C a string containing the name of the data to be written,
C as it appears in the XML description
name
```

Output parameter

None.

Example

```
CALL tdt_fwrite (tdtstate, astruct, "astruct");
```

3.2.4 tdt_fread()

Purpose

To read data from the connection given by the `TDTState` parameter as per the XML identifer string (parameter `"name"`).

Input parameters

```
C The identifier for this data transfer
tdtstate
```

```
C the data being read
val

C a string containing the name of the data to be read,
C as it appears in the XML description
name
```

Output parameter

None.

Example

```
CALL tdt_fread (tdtstate, astruct, "astruct");
```

3.2.5 tdt_fclose()

Purpose

Closes the connection or open files and frees TDT-allocated memory.

Input parameters

```
C The identifier of the connection to be closed
ts
```

Output parameter

None.

Example

```
CALL tdt_fclose (tdtstate);
```

3.2.6 tdt_fend()

Purpose

Frees memory allocated by the TDT for the configuration information.

Input parameters

```
C The identifier of the stored configuration data
tc
```

Output parameter

None.

Example

```
CALL tdt_fend (tdtconf);
```

3.3 Compiling and linking

3.3.1 Compiling the library

The TDT can be compiled as a library, `libtdt.a`. This must be in the library path of the development environment, or a directory specified by the `-L` flag in gcc. A makefile is included with the source files. Running '`make lib`' from within the `tdt` source directory will rebuild the TDT library.

3.3.2 Linking with your own programs

First build the library `libtdt.a` as explained above.

Compile your application...

```
xlf -c -Wall <program>.f
```

... and link:

```
xlf <program>.o -L../tdt -o <program> -ltdt -lexpat_linux \
    -o <program>
```

where `<program>.o` is the output from the compilation of `<program>.f`

This assumes that the Expat XML parser library is available system-wide on your machine. If it is not, and you want to use the Expat library supplied with the TDT, use the following link command:

```
xlf <program>.o -L<location of TDT library> -ltdt /
                -L<location of Expat library> -lexpat
```

3.4 An example program

```
1  PROGRAM fclnt
2  INTEGER:: data1(2,5)
3  REAL*8:: data2(10,15,10,10)
4  INTEGER tdtstate1
5  INTEGER tdtstate2
6  INTEGER tdtconf
7
8  DO k = 1,2
9      DO l = 1,5
10     data1(k,l) = k*10 + l
11     PRINT *, data1(k, l)
```

```
12      END DO
13  END DO
14
15  CALL tdt_fconfigure (tdtconf, 'clntconf.xml')
16
17  CALL tdt_fopen (tdtstate1, tdtconf, 'conn1')
18  CALL tdt_fwrite (tdtstate1, 'a', data1)
19
20  CALL tdt_fopen (tdtstate2, tdtconf, 'conn2')
21  CALL tdt_fread (tdtstate2, 'b', data2)
22
23  PRINT *, 'I read:'
24  PRINT *, data2
25
26  CALL tdt_fclose (tdtstate1)
27  CALL tdt_fclose (tdtstate2)
28  CALL tdt_fend (tdtconf)
29
30  END
```

The TDT-specific lines are the following:

4, 5: Declare two `TDTStates`. One is for a `tdt_fread` and the other for a `tdt_fwrite`.

6: Declare a `TDTConfig` variable.

15: Parse the configuration data

17: Open the first connection

18: Write data on the open connection

20: Open another connection

21: Read data from the new connection

26, 27, 28: Close connections and tidy up.

3.5 Error conditions

The user functions of the TDT will crash, returning with error code 1, if an error occurs. It will also output a debugging message to `stderr`.

4 TDT Data descriptions in XML

XML (eXtensible Markup Language) is a widely used format for structured data and documents. For a more detailed introduction to XML, see

http://www.w3schools.com/xml/default.asp

4.1 <data_desc> tags

This is the root element (or document element) required by XML.

Attributes : none.

4.2 <decl> tag

Each XML description of a model is made up of one or more variable declarations, which are identified by `<decl>` tags.

Attributes : `name="abcde"` where `abcde` is one or more alphanumeric characters.

4.3 <struct> tag

The `<struct>` and `</struct>` tags surround one or more declarations, analogous to a `struct` declaration in C.

Attributes : none.

4.4 <array> tag

The `<array>` and `</array>` tags surround a text element indicating the primitive type, or a declaration if the array is made up of composite types.

Attributes : `size="n"` where n is an integer. If the size of the array is unknown at the time the datadesc is being created, it can be set to zero and later given a size with the `tdt_size_array()` function.

4.5 <addr> tag

The `<addr>` and `</addr>` tags surround declarations, indicating that this is a pointer to a value, not the value itself.

Attributes : none.

4.6 Text elements

Text elements are the value which appear between start and end tags. For TDT XML, text elements are used to indicate the primitive data type of a declaration (`<decl>`) or an array (`<array>`).

The allowed values are `"int"`, `"double"`, `"float"`, and `"char"`.

4.7 Example

The following XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<data_desc>

    <decl name="vardouble">double</decl>

    <decl name="astruct">
        <struct>
            <decl name="elem1">
                    <array size="2">int</array></decl>
            <decl name="elem2">double</decl>
        </struct>
    </decl>

    <decl name="varint">int</decl>

    <decl name="dyn">
        <array size="0">double</array>
    </decl>

</data_desc>
```

represents describes the data being transferred in the client program above. It describes the following C declarations:

```
double vardouble;

struct {
    int    elem1[2];
    double elem2;
} astruct;

int varint;
double *dynarray;
```

5 TDT Configuration files

The configuration files used by TDT are also written in XML.

5.1 <program> tag

The document-level tag for a configuration file is the `<program>` tag.

Attributes: `name="abcde"`, where `"abcde"` is an alphanumeric string.

5.2 <channel> tag

The `<channel>` tag specifies all the information required to establish a TDT connection and send or receive the data in the correct format.

Attributes:

`name` : the name by which the channel will be referenced in the code.

`mode` : the mode of the channel, i.e. input or output (`"in" | "out"`)

`type` : the type of the connection: `"socket" | "file"`

`host` : if type is `"socket"`, the hostname to use

`port` : if type is `"socket"`, the port to use

`filename` : if type is `"file"`, the filename to read/write from/to

`datadesc` : the name of the datadesc XML which describes the data that will pass on this channel

5.3 Example configuration file

```
<program name="client">
  <channel name="clnt_to_serv"
           mode="out"
           host="localhost"
           port="2222"
           type="socket"
           datadesc="datadesc.xml">
  </channel>
</program>
```

This configuration describes a program called `"client"` which has one communication channel, namely output socket 2222 from machine `"localhost"`. The datadesc for this connection is in the file `"datadesc.xml"`. Note that the attribute `"filename"` is not required in this example.

6 Sample applications

The sample applications are provided with Makefiles in order to compile and link the programs.

6.1 C Examples

The sample C programs (`testclnt.c` and `testserv.c` in the `tests` subdirectory of the TDT source directory) demonstrate writing data from a client application (`testclnt`) to a server application (`testserv`). `testclnt` writes three variables (a `double`, a `struct` containing an array of `int` and a `double`, and an `int`) to `testserv`. In this case, only one `TDTState` (for one XML description and one communication channel) is required.

Building the examples

Type `"make all"` from within the `tdt/tests` directory.

In `tdt/tests` this will create the programs `"serv"` and `"clnt"`.

6.2 Fortran examples

The sample Fortran applications are two programs (`prog1.f` and `prog2.f` in the `ftdt/tests` subdirectory) which read and write data to/from each other. `prog1` writes a 2x5 matrix of INTEGERs for `prog2` to read and reads a 5x2 matrix of doubles (REAL*8) which is written by `prog2`.

Two XML files are provided (`"example.xml"` and `"example2.xml"`). These describe the two matrices. In these examples the XML files are expected to be in the same directory as the programs reading them.

Two `TDTState` variables are used by each program (`tdtstate1` and `tdtstate2`). Each `TDTState` contains information about one matrix and the channel by which that data will be transferred.

Building the examples

Type `"make all"` from within the `ftdt/tests` directory.

In `ftdt/tests`, the programs will be called `"prog1"` and `"prog2"`.

7 "make" parameters

The following parameters can be passed to `make`:

`FF` : specifies the name of the Fortran compiler (default is `xlf`).

`EXPAT` : specifies the location of the Expat libraries (default is `../../expat`).

`OS` : specifies the operating system (Linux or AIX). This is used to select the appropriate Expat library (default is Linux).

Example:

```
make FF=g77 EXPAT=~/expat1.95.2 OS=AIX all
```

`"make clean"` will remove old object (`*.o`) files and executable programs from the `tdt/tests` and `tdt` directories (or `ftdt/tests` and `ftdt`).

8 Further information

If you need further information, have comments or requests for enhancements, or you've found a bug, you can contact the authors:

Ciaron Linstead <linstead@pik-potsdam.de>

Cezar Ionescu <ionescu@pik-potsdam.de>

Dan Beli <beli@pik-potsdam.de>