TMlib User Guide

J. Lynch, University of Minnesota
J. Silvis, University of Minnesota
K. Goetz, University of Minnesota

22nd November, 2006

Table of Contents

1. Introduction

This document describes the TMlib client software library. Included is a description of the darchitecture of the library, installation instructions for various platforms, example applications including source code, an API reference guide with error codes, and a list of common errors and questions regarding the usage of the library.

Current versions of this document and the TMlib client software library can be found online at the S/WAVES download page [http://homepage.mac.com/swaves].

2. TMlib Overview

2.1. Concepts

2.1.1. Mission / Spacecraft / Instrument / EventType (MSIE)

Data in the TMlib model is organized in an inverted tree structure, much like the structure of the filesystem on a modern operating system. The user first needs to know the Mission for which they would like to retrieve data. Next, in the case of the STEREO mission, there are two Spacecraft, as such, the user must select one. After selecting the spacecraft, an Instrument on the spacecraft must be selected to gather events from. In our example below we've chosen the SWAVES instrument. Finally, after choosing the instrument, the user selects an EventType. Each EventType has its own unique set of items that it contains, though all items from event levels above the chosen one are also valid for the current EventType. For example, an LFR EventType would contain all items specific to itself along with any items that are valid for any upper levels, such as CCSDS items which are valid for all EventTypes. See the online help system for a better understanding of what this means.

When connecting to the TMlib server a user needs to specify the Mission, the Spacecraft, the Instrument, and the EventType to the library. This quadruple is called the MSIE domain. The MSIE domain is used to specify a set of data the user is interested in getting as mentioned above. For example, if a user wanted to see every packet coming from the spacecraft, they would specify an MSIE domain such as this:

```
/* Example in C */
/*                             Mission       Spacecraft      Instrument      EventType
*/
TM_Select_Domain(&stream_id,   "stereo",     "stereo_a",      "swaves",      "packet");
```

Note

To find the list of valid missions, spacecraft, instruments, events, and items, go to http://<your.tmlib.server.name.or.ip>:8080/TMlib/

The `stream_id` variable in the above code fragment is a unique identifier or handle that TMlib returns to the user program for use in subsequent TMlib function calls, it is explained in the detailed API documentation in this book. The use of a session identifier allows a user program to have multiple open connections to the server simultaneously.

Caution

That said, the TMlib client software library is not thread safe. Opening multiple connections in mutiple threads can (and probably will) result in undefined behavior.

2.1.2. Streams, Events, and Items

After specifying the MSIE, the user has to specify a stream. A stream is a data source; a place from which the user would like to get data. For example, the stream could be the realtime stream coming from the spacecraft, it could be a path to a file on a TMlib server containing a particular day's worth of recorded data, or it could be the chooser. Though the chooser is not technically a stream, it opens a file browser that allows the user to select a stream. The chooser is described in the following sections. Some examples of selecting a stream:

```
; Example in IDL
TM_Select_Stream_File(stream_id, 'realtime')
```

```
; or
TM_Select_Stream_File(stream_id, '+/path/to/file/on/server') ; note the preceding plus
'+' sign
; or
TM_Select_Stream_File(stream_id, 'chooser')
; or
TM_Select_Stream_Timerange(stream_id, startUR8, endUR8)
```

Note

The realtime stream may or may not exist on the TMlib server you use. The realtime stream will be deprecated after launch.

After the domain and the stream have been specified, the user calls the event function and waits for the requested event to be built, for example:

```
// Example in Java
import edu.space.umn.swaves.*;

...

for (;;) {
TM.TM_Find_Event(stream_id);

...
}
```

The `TM_Find_Event` function will block and wait for the event to be built on the server. After the event is complete, the server will notify the client and the function will return. The user can then call any of the `TM_Get_Item` functions appropriate for the EventTypes they have requested. Complete examples are located in Section 5, though here is a quick example to illustrate this point:

```
/* Example in C */
#define ITEM_BUFFER_SIZE 1

int number_of_returned_elements, item_buffer[ITEM_BUFFER_SIZE];

...

TM_Get_Item_I4(stream_id, "CCSDS_APID", item_buffer, ITEM_BUFFER_SIZE,
&number_of_returned_elements);

printf("APID: %d\n", item_buffer);

...
```

In the above code snippet, we ask the server for the CCSDS_APID item (from the "/stereo/stereo_a/swaves/packet" MSIE). We know that the item we have requested, CCSDS_APID, is a scalar of size 1 (that is, one "I4" [or four 8bit bytes]), so `item_buffer` only needs to be of size 1. Also note that the number of elements in the returned array is stored in the variable `number_of_returned_elements`. By the way, if we expected a vector back from TMlib but had not given it enough storage in the array we passed in, the `TM_Get_Item_I4` function would return a fatal (negative) error stating that the server returned more data than space available in the array. The contents of the array passed into the function would be invalid and of course should not be used.

Note

Always check the return values for TMlib client functions!

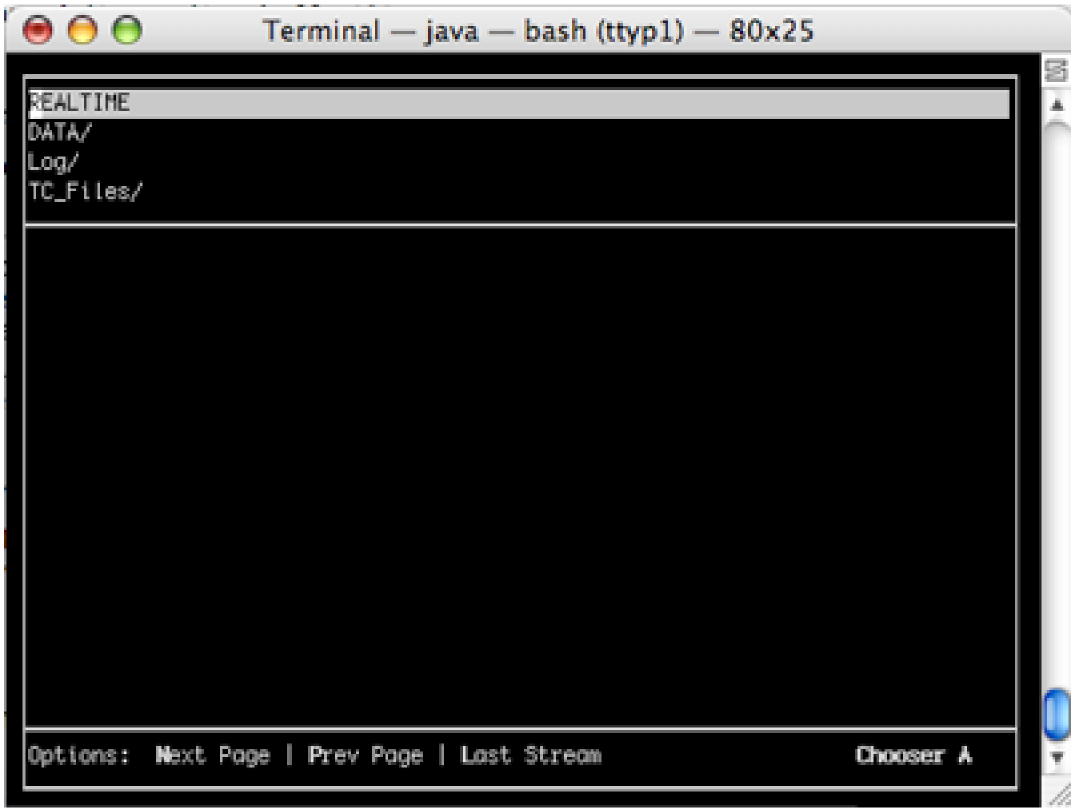
2.1.3. The Chooser

As noted above, the chooser is an application embedded in TMlib. It allows the user to easily browse through the file directory hierarchy on the server and choose data files, or from the realtime stream if so desired.

2.1.3.1. Mac OS X

When a user specifies "chooser" for the stream argument to `TM_Select_Stream`, a small application embedded into the TMlib client will open in the current terminal session. From here the user can navigate the directory hierarchy to find the file containing the data they are interested in. The screenshot below is of the root level. The user can also press the 'L' key to automatically select the last data file or stream selected. You can see the shortcut in the bottom of the screenshot below.

Figure 1. Root level of the Mac OS X curses-based chooser



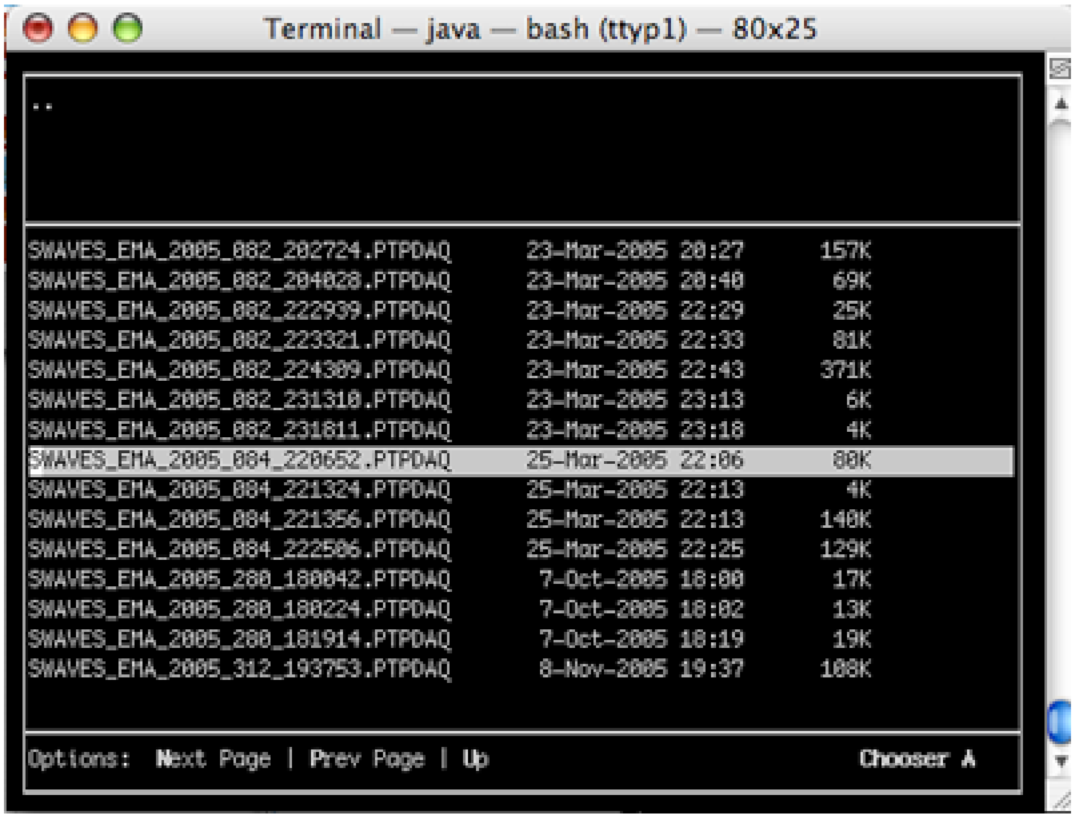
```
REALTIME
DATA/
Log/
TC_Files/

Options:  Next Page | Prev Page | Last Stream                  Chooser A
```

Navigating via the arrow keys to "DATA" and pressing enter, you will find the hierarchy starting with the mission. Also note that in any screenshots below the chooser root level, there is an item called "..". Selecting this item will bring the user back to the previous level, functioning identically to a parent directory.

After navigating to the level containing the data files or stream desired, use the arrow keys to move to the file to select it and press enter. This will close the chooser window and tell the server to start streaming the data from this file to the user's application. From this point on, control is given back to the user's application. A screenshot of selecting a file is included below.

Figure 2. Selecting a data file in the chooser.



```
..

SWAVES_EMA_2005_082_202724.PTPDAQ      23-Mar-2005 20:27      157K
SWAVES_EMA_2005_082_204028.PTPDAQ      23-Mar-2005 20:40       69K
SWAVES_EMA_2005_082_222939.PTPDAQ      23-Mar-2005 22:29       25K
SWAVES_EMA_2005_082_223321.PTPDAQ      23-Mar-2005 22:33       81K
SWAVES_EMA_2005_082_224309.PTPDAQ      23-Mar-2005 22:43      371K
SWAVES_EMA_2005_082_231310.PTPDAQ      23-Mar-2005 23:13        6K
SWAVES_EMA_2005_082_231811.PTPDAQ      23-Mar-2005 23:18        4K
SWAVES_EMA_2005_084_220652.PTPDAQ      25-Mar-2005 22:06       80K
SWAVES_EMA_2005_084_221324.PTPDAQ      25-Mar-2005 22:13        4K
SWAVES_EMA_2005_084_221356.PTPDAQ      25-Mar-2005 22:13      140K
SWAVES_EMA_2005_084_222506.PTPDAQ      25-Mar-2005 22:25      129K
SWAVES_EMA_2005_280_180042.PTPDAQ       7-Oct-2005 18:00       17K
SWAVES_EMA_2005_280_180224.PTPDAQ       7-Oct-2005 18:02       13K
SWAVES_EMA_2005_280_181914.PTPDAQ       7-Oct-2005 18:19       19K
SWAVES_EMA_2005_312_193753.PTPDAQ       8-Nov-2005 19:37      108K

Options:  Next Page | Prev Page | Up                    Chooser A
```

The file selected will be saved in the `.tmlibrc` file in the user's home directory in case the user would like the invoke the "Last Stream" command (the 'L' key) at the chooser's root level. In this case we are about to select `SWAVES_EMA_2005_084_220652.PTPDAQ`. We found this file by descending down through "DATA"->"STEREO"->"STEREO_A"->"SWAVES".

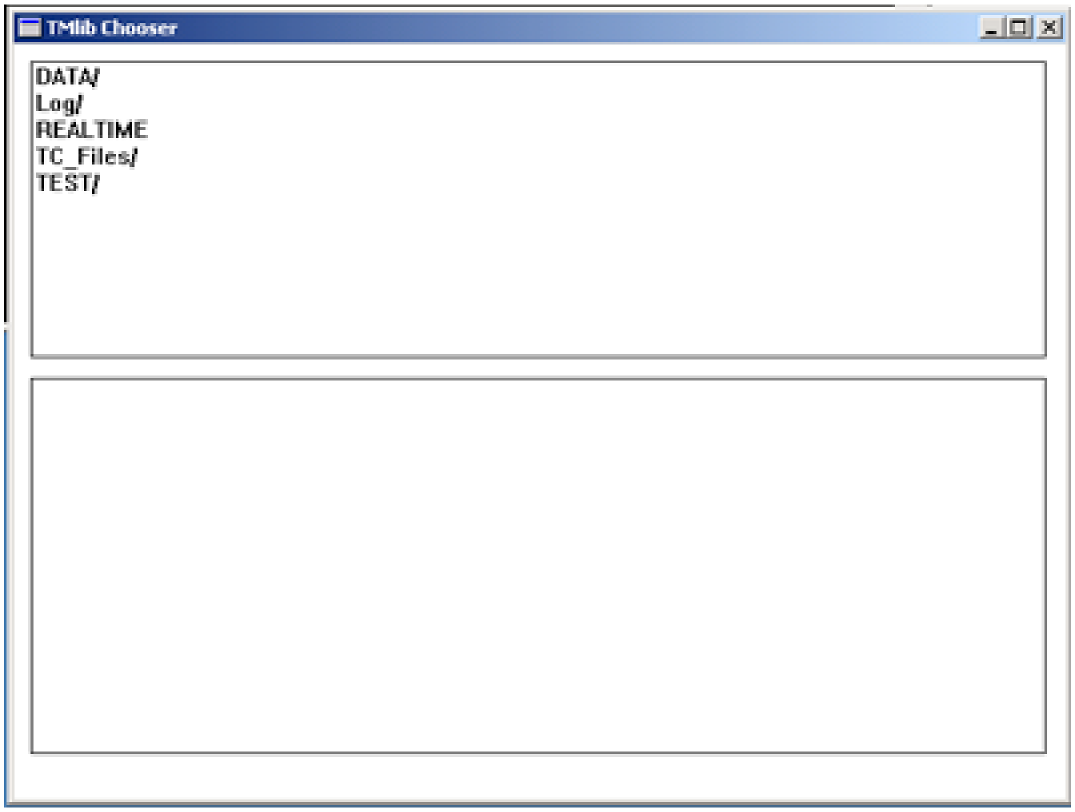
Note

The filename itself has valuable information encoded into it. Starting from the left, and delimited by the underscore character, the filename contains the instrument name, the spacecraft name, the year, the day of year, and the time (UTC).

2.1.3.2. Windows XP

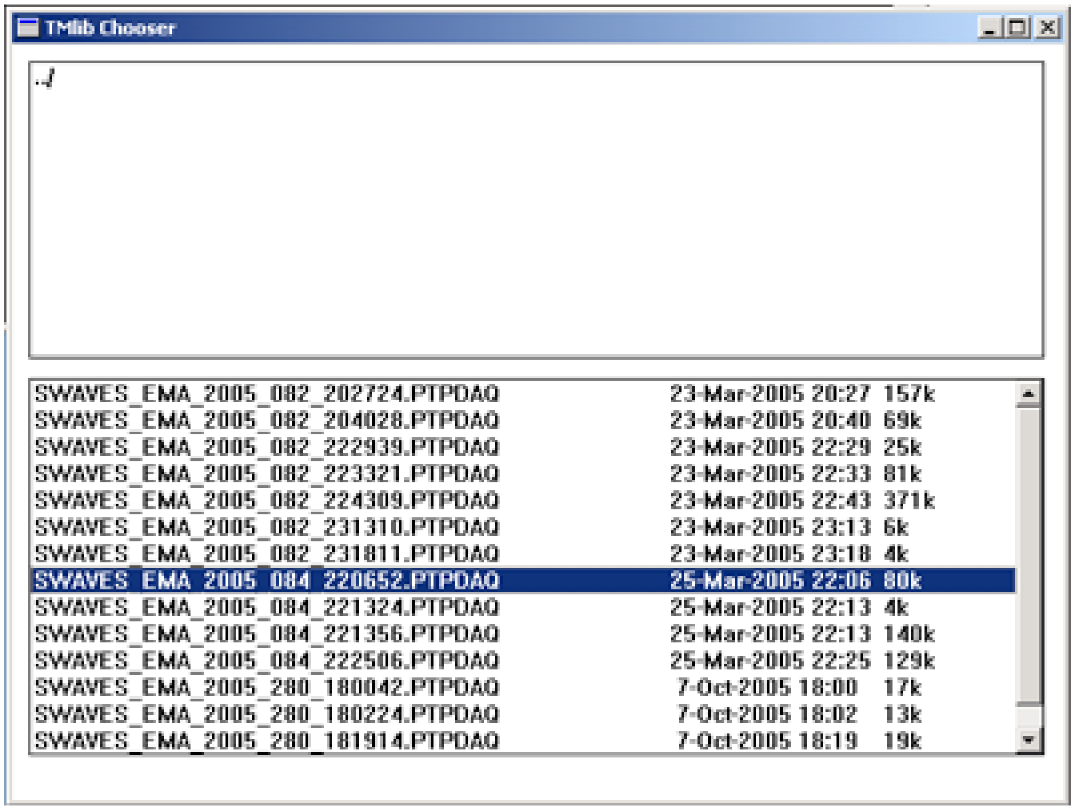
The Windows based chooser functions exactly the same way the Mac OS chooser functions with the exception that instead of a terminal-based application taking over a console window, a normal window will open up a screen much like that of the chooser on Mac OS X. The user can navigate the chooser by double-clicking. A screenshot of the Windows chooser at its root level is located below.

Figure 3. Windows chooser



Under the Windows chooser, the "Last Stream Chosen" functionality is implemented by the "Last Stream Chosen" item in the files pane of the Windows chooser. Below is a screenshot of a user selecting a data file to be read.

Figure 4. Selecting a file via the Windows chooser



```
../
```

```
SWAVES_EMA_2005_082_202724.PTPDAQ      23-Mar-2005 20:27  157k
SWAVES_EMA_2005_082_204028.PTPDAQ      23-Mar-2005 20:40  69k
SWAVES_EMA_2005_082_222939.PTPDAQ      23-Mar-2005 22:29  25k
SWAVES_EMA_2005_082_223321.PTPDAQ      23-Mar-2005 22:33  81k
SWAVES_EMA_2005_082_224309.PTPDAQ      23-Mar-2005 22:43  371k
SWAVES_EMA_2005_082_231310.PTPDAQ      23-Mar-2005 23:13  6k
SWAVES_EMA_2005_082_231811.PTPDAQ      23-Mar-2005 23:18  4k
SWAVES_EMA_2005_084_220652.PTPDAQ      25-Mar-2005 22:06  80k
SWAVES_EMA_2005_084_221324.PTPDAQ      25-Mar-2005 22:13  4k
SWAVES_EMA_2005_084_221356.PTPDAQ      25-Mar-2005 22:13  140k
SWAVES_EMA_2005_084_222506.PTPDAQ      25-Mar-2005 22:25  129k
SWAVES_EMA_2005_280_180042.PTPDAQ       7-Oct-2005 18:00  17k
SWAVES_EMA_2005_280_180224.PTPDAQ       7-Oct-2005 18:02  13k
SWAVES_EMA_2005_280_181914.PTPDAQ       7-Oct-2005 18:19  19k
```

Again notice the presence of the "../" item in the above screenshot. As in Mac OS X, this item serves to bring the user back to the previous level if selected.

2.1.4. API Overview

The table below lists all the functions that comprise the item retrieval portion of TMlib. The exact signatures of these functions are listed in a later section. For now, these tables are a brief introduction and reference to the TMlib API.

Table 1. TMlib Item API

Function Name	Description
`TM_Select_Server`	(optional) Selects the TMlib server to connect to. Overrides the server parameter in the currently selected site.
`TM_Select_Port`	(optional) Select the TCP port on the TMlib server to connect to. Rarely used. Overrides the port parameter in the currently selected site.
`TM_Select_Domain`	Connect to a TMlib server using the specified MSIE.
`TM_Select_Stream_File`	Selects a particular stream valid for the MSIE domain. Valid arguments are: "realtime", "<filename>", "chooser".
`TM_Select_Stream_TimeRange`	Selects a time range for the stream. A start and end time, in UR8, are required.
`TM_Find_Event`	Blocks until the next valid event is built on the selected stream.
`TM_Get_Item_I4`	Once an event is ready, retrieves the value of an item as a 32bit integer scalar or vector.
`TM_Get_Item_R4`	Once an event is ready, retrieves the value of an item as a 32bit floating point scalar or vector.
`TM_Get_Item_R8`	Once an event is ready, retrieves the value of an item as a 64bit floating point scalar or vector.
`TM_Get_Item_Char`	Once an event is ready, retrieves the value of an item as an 8bit integer scalar or vector.
`TM_Set_Position`	Takes a UR8 and positions the current time position in the stream to the specified value.
`TM_Get_Position`	Returns the current time position in the stream as a UR8. See section 4.2.
`TM_Error_Stack`	If any of the above functions return an error, this function can be called to get more information about the error.
`TM_Close`	Close the connection to the TMlib server gracefully.

In addition to the aforementioned item retrieval functions, there are a host of convenience functions that handle dates and times and the conversions between them. The table below lists all of these remaining functions. Again, the exact signatures of these functions are described in Section 4.2.

Table 2. TMlib Time API

Function Name	Description
`TM_UR8_to_String_Absolute`	Converts a UR8 to a string of the format "18-Sep-1983:11:42:25.005".
`TM_UR8_from_String_Absolute`	Converts a date string of the format "18-Sep-1983:11:42:25.005" to UR8 format.
`TM_UR8_to_YDOYH`	Takes a UR8 and returns three integers representing the year, the day of year, and the fractional hour.
`TM_UR8_to_YDOY`	Takes a UR8 and returns three integers representing the year, the day of year, and the microseconds.
`TM_UR8_from_YDOY`	Takes three integers representing the year, the day of year, and the microseconds and returns a UR8
`TM_UR8_to_YMD`	Takes a UR8 and returns 7 integers representing the year, month, day, hour, minute, second, and microsecond.
`TM_UR8_from_YMD`	Takes 7 integers representing the year, month, day, hour, minute, second, and microsecond and returns a UR8
`TM_UR8_to_String_Comparison`	Converts a UR8 to a string of the format "1983-09-18 11:42:25.056".
`TM_UR8_from_String_Comparison`	Converts a string of the format "1983-09-18 11:42:25.056" to a UR8
`TM_UR8_to_String`	Takes a UR8 and returns a string of the format "18-Sep-1983, 11:42:25.056"
`TM_UR8_from_localtime`	Takes the current time on the user's computer and converts it into a UR8.

2.2. TMlib Design

The TMlib client software library is at the lowest level a core C library with a binding for each language that it supports. Each language binding uses its own methodology to utilize the functions present in the core C library. For instance, to reference functions in the C library, Java uses JNI, and IDL uses CALL_EXTERNAL.

A simple architectural diagram is listed below. The TMlib software library comprises the lower three levels.

User Applications			
IDL	Java	C	Fortran
C/IDL Bridge	C/Java Bridge	C/C Bridge	C/Fortran Bridge
C Library			

As you can see, an application can use any language binding to access functions present in the C library. In this model, application programming interfaces are kept consistent across languages and host operating systems.

Caution

Again, the TMlib client software library is not thread safe. Opening multiple connections in mu-tiple threads can (and probably will) result in undefined behavior.

3. Installation, Configuration, and Verification

This section describes how to install the TMlib client on supported platforms, how to configure TMlib on supported platforms and how to verify that the software was installed and configured correctly.

3.1. Installation

3.1.1. Mac OS X

TMlib is distributed in PackageMaker form for Mac OS X. Installation is straight forward and consists of running the package installer and following the prompts. After the package installer completes, the TMlib library will be installed in the `/Applications/STEREO/TMlib_Client` folder of your main hard drive. In that directory you will find example code, startup files, utility programs, documentation, and the client libraries themselves.

Note

After installation, your `.tmlibrc` file will have been copied to `.tmlibrc.bak` and a new `.tmlibrc` file will have been copied in its place with the current list of servers. If you have local modifications to this file, they will have to be merged manually.

As part of the installation process, the installer will modify your `.cshrc` and/or `.bash_profile` scripts to execute a startup file located in the TMlib installation directory called `TMlib_Startup.sh` or `TMlib_Startup.csh` depending on which shell you use. This file initializes some environment variables which are described below.

After installation has completed you should log out and log back in again to allow the environment variables to be set correctly. Advanced users could also do this by hand by manually sourcing their shell initialization scripts.

3.1.2. Windows XP

TMlib for Windows is distributed in MSI package form.

Note

After installation, your `tmlibrc` file will have been copied to `tmlibrc.bak` and a new `tmlibrc` file will have been copied in its place with the current list of servers. If you have local modifications to this file, they will have to be merged manually.

Note

Older versions of Windows (pre-Windows 2000 Service Pack 3) may need to install the newest version of the Windows installer from Microsoft's website. If you are having trouble installing the software, this may be the issue.

Note

Though it is assumed that TMlib will function correctly on Windows platforms as far back as Windows 95, the oldest release of Windows officially supported is Windows 2000 Service Pack 3.

Like Mac OS X, installation on the Windows platform is very straight forward. Run the installer and follow the prompts. If you've installed TMlib in the default directory it will be located in `C:\Program`

`Files\STEREO\TMlib_Client`. Again, under that directory is where you will find all the examples, client utilities, etc.

Under Windows, all the same environment variables are set as they are in Mac OS X except of course that there is no shell script to set them. Instead, all of the environment variables have user scope and are set via the standard Windows mechanism which is described below.

After installation has completed you should log out and log back in again to allow the environment variables to be set correctly.

3.2. Configuration

TMlib is normally configured via a set of environment variables and via the configuration utility called **tmconfig**. They are described below.

3.2.1. Environment Variables

As noted in the table below, STEREO_TMlib_Client_Home is the only TMlib client specific environment variable. The rest of the variables listed are development environment specific.

Table 3. TMlib Environment Variables

`STEREO_TMlib_Client_Home`	Contains the path to the TMlib client installation. This variable is absolutely critical to the proper operation of TMlib. It is set by the installer.
`CLASSPATH`	Contains the Java classpath. Though not specific to TMlib the installation program appends the path to the JAR file containing the TMlib routines to this.
`PATH`	Contains the directory search list for executables. Though not specific to TMlib, the installer modifies this variable and adds the path to the TMlib utility applications.
`IDL_PATH`	The IDL search path. Again, though not specific to TMlib, the installer modifies this variable to contain the path to the file containing the TMlib IDL routines.
`IDL_STARTUP`	Contains the IDL script file to execute when IDL starts up. This is not used under Unix-like platforms since a shell alias called **sidl** is created by the installer.

3.2.2. tmconfig

The TMlib client now has a new way to configure and specify parameters called **tmconfig** that replaces the old (and deprecated) environment variable method.

Note

Though **tmconfig** is the recommended way of configuring TMlib, the old environment variable method is also supported for backward compatibility, thus `STEREO_Server` and `STEREO_TM` can still be used to specify the TMlib server and port, respectively.

Note

The aforementioned environment variables will take precedence over the values in the `.tmlibrc` configuration file.

tmconfig is a command line utility that can be used to maintain what are called "sites". Within each site is contained a set of parameters that can uniquely identify a site. Parameters such as server name or IP, server port number, last file chosen in the chooser, and what the current default site are all available via this command. For example, to list all the available sites you use the -l (ell) option:

```
unix_or_windows$ tmconfig -l
RT TMserver A at UMN           128.101.218.170
RT TMserver A at APL           128.244.181.238
RT TMserver B at APL           128.244.181.239
RT TMserver at Meudon          145.238.10.253
TMserver at GSFC               128.183.134.187
TMserver at UCB                128.32.147.148
localhost                      127.0.0.1       (DEFAULT)
```

Notice how the listing provides the symbolic name of the server and its IP address. Also note the item marked as "DEFAULT". This is the site that would be used for any TMlib client programs, it functions like with the old `STEREO_Server` environment variable. To select another site for use, the -d and -s options are utilized:

```
unix_or_windows$ tmconfig -d -s umn
unix_or_windows$ tmconfig -l
RT TMserver A at UMN           128.101.218.170 (DEFAULT)
RT TMserver A at APL           128.244.181.238
RT TMserver B at APL           128.244.181.239
RT TMserver at Meudon          145.238.10.253
TMserver at GSFC               128.183.134.187
TMserver at UCB                128.32.147.148
localhost                      127.0.0.1
```

Note how only a few characters of the site name we wanted to become default were specified. In this way, descriptive names can be used, but the user is not required to type the entire site name to specify it, only a few unique characters need be used.

Sites can be added, modified, and deleted via the command-line utility or by manually modifying the configuration file itself. If modifying the configuration file manually, care must be taken to maintain a valid XML document. The file is located in the user's home directory on Unix-like platforms and is named `.tmlibrc` or on the Windows platform it is located in the `STEREO_TMlib_Client_Home` directory and is named `tmlibrc`. Comments are provided in the configuration file itself on what all the valid XML elements are and how they are used. An example configuration file is listed below:

Modifying the file via the command line utility **tmconfig** is relatively straightforward as well and can be accomplished by reading the command help which is available via the -h option:

```
unix_or_windows$ tmconfig -h
usage: tmconfig [-d|-l|-c|-a|-r|-m {key=value}] [-s {sitename}]

Only one command at a time may be issued. Enclose parameters containing
spaces in quotes.

Commands:
-d              Set a site to be the default site. Requires -s
-l              List all available sites.
-c              Create a new configuration file. OVERWRITES EXISTING FILE!!!
-a              Add a site. Requires -s
-r              Remove a site. Requires -s
-m key=value    Modify specfied key=value pair. Without -s, operates on DEFAULT
```

```
site. Valid keys are: NAME, SERVER, LASTFILE, and PORT

Options:
-s sitename     Specify site name.
```

The normal steps in creating a site consist of adding a site and then modifying the appropriate key/value pairs to tailor it to personal preferences.

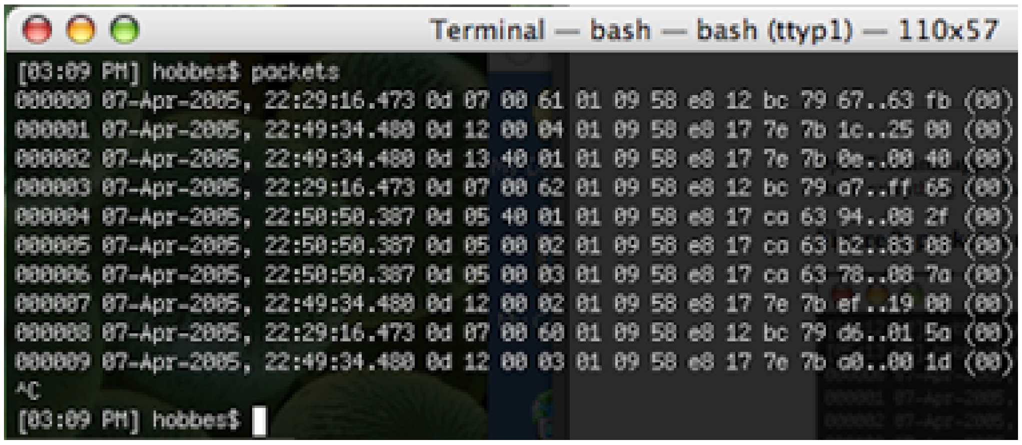
3.3. Verification

To verify the installation make sure to log out and log back in again on both platforms. This ensures that all the proper environment variables will be set. Also, double check that the STEREO_Server environment variable is pointed at a valid TMlib server.

3.3.1. Mac OS X

Open **Terminal.app** (this application is located under `/Applications/Utilities`) and type **packets**. If everything is installed correctly, after a short delay you should start seeing output of this form:

Figure 5. packets output



```
[03:09 PM] hobbes$ packets
000000 07-Apr-2005, 22:29:16.473 0d 07 00 61 01 09 58 e8 12 bc 79 67..63 fb (00)
000001 07-Apr-2005, 22:49:34.480 0d 12 00 04 01 09 58 e8 17 7e 7b 1c..25 00 (00)
000002 07-Apr-2005, 22:49:34.480 0d 13 40 01 01 09 58 e8 17 7e 7b 0e..00 40 (00)
000003 07-Apr-2005, 22:29:16.473 0d 07 00 62 01 09 58 e8 12 bc 79 a7..ff 65 (00)
000004 07-Apr-2005, 22:50:50.387 0d 05 40 01 01 09 58 e8 17 ca 63 94..08 2f (00)
000005 07-Apr-2005, 22:50:50.387 0d 05 00 02 01 09 58 e8 17 ca 63 b2..63 08 (00)
000006 07-Apr-2005, 22:50:50.387 0d 05 00 03 01 09 58 e8 17 ca 63 78..08 7a (00)
000007 07-Apr-2005, 22:49:34.480 0d 12 00 02 01 09 58 e8 17 7e 7b ef..19 00 (00)
000008 07-Apr-2005, 22:29:16.473 0d 07 00 60 01 09 58 e8 12 bc 79 d6..01 5a (00)
000009 07-Apr-2005, 22:49:34.480 0d 12 00 03 01 09 58 e8 17 7e 7b a0..00 1d (00)
^C
[03:09 PM] hobbes$
```

The **packets** program also supports the "-c" option which will bring up the chooser and allow you to choose a stream other than realtime, which **packets** uses by default. This can be useful if you are not connected to a server with a realtime data stream as would be the case after spacecraft launch.

3.3.2. Windows XP

Under Windows the steps are almost identical. Open up a Command window but going to Start->Run, type **cmd**, and click Run. Then type **packets** and you should see output that is almost identical to the image above. As under Mac OS X, the **packets** program supports the "-c" option for bringing up the chooser and selecting a stream.

4. API Reference

Included below is a formal description of the TMlib application programming interface.

4.1. Item Functions

Described herein are all the API calls available to the user via the TMlib API. Each subsection contains a different API that describes its function, what it returns, and what its parameters are. The function parameters are described in order from left to right. An example of each function's signature is also given in all supported language bindings.

Note

In the C prototypes below, note the use of the typedef'd types such int32_t and uint32_t. These are available to ensure the user is indeed using a 32bit integer. They are imported (or defined in the case of Windows) for use by the `libTM.h` header.

Note

Always check the return values for TMlib client functions!

4.1.1. `TM_Select_Domain`

Description: Selects the MSIE. Called first when setting up a connection.

Returns: Non-zero integer on failure.

Arguments	Description
stream_id	An empty buffer for storing the returned unique stream identifier.
mission	String that contains the Mission name.
spacecraft	String that contains the Spacecraft name.
instrument	String that contains the Instrument name.
event	String that contains the EventType.

```
/* C */
#include "libTM.h"

int32_t TM_Select_Domain(uint32_t *stream_id, char *mission, char *spacecraft, char
*instrument, char *event);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_Select_Domain(IntBuffer stream_id, String mission, String spacecraft, String
instrument, String event);
```

```
; IDL

LONG TM_Select_Domain(LONG stream_id, STRING mission, STRING spacecraft, STRING
instrument, STRING event)
```

4.1.2. `TM_Select_Stream_File`

Description: Selects the data source. Called after the MSIE has been selected.

Returns: Non-zero integer on failure.

Arguments	Description
stream_id	The returned unique stream identifier.
stream	A string for choosing a stream. This argument can be one of "realtime", "chooser", or an explicit path-name to a data file residing on the server such as "+/path/to/file" (note the '+' prefix).

```
/* C */
#include "libTM.h"

int32_t TM_Select_Stream_File(uint32_t streamid, char *stream);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_Select_Stream_File(int streamid, String stream);
```

```
; IDL

LONG TM_Select_Stream_File(LONG streamid, STRING stream);
```

4.1.3. `TM_Select_Stream_TimeRange`

Description: Selects the data source designated by a start and end time. Called after the MSIE has been selected. Useful when data sets are large or events cross into the next day's worth of data. A startUR8 that is -1 means Beginning of Information (BOI). An endUR8 that is zero means End of Day (EOD) and an endUR8 that is -1 means End of Information (EOI).

Returns: Non-zero integer on failure.

Arguments	Description
stream_id	The returned unique stream identifier.
startUR8	A double precision floating point number, in UR8 format, that defines the start time in the server's data set.
endUR8	A double precision floating point number, in UR8 format, that defines the end time in the server's data set.

```
/* C */
#include "libTM.h"

int32_t TM_Select_Stream_TimeRange(uint32_t streamid, double startUR8, double endUR8);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_Select_Stream_TimeRange(int streamid, double startUR8, double endUR8);
```

```
; IDL

LONG TM_Select_Stream_TimeRange(LONG streamid, DOUBLE startUR8, DOUBLE endUR8);
```

4.1.4. `TM_Find_Event`

Description: Blocks until an event is built on the server. Called after selecting a stream.

Returns: Non-zero integer on failure.

Arguments	Description
stream_id	The returned unique stream identifier.

```
/* C */
#include "libTM.h"

int32_t TM_Find_Event(uint32_t streamid);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_Find_Event(int streamid);
```

```
; IDL

LONG TM_Find_Event(LONG streamid)
```

4.1.5. `TM_Get_Item_I4`

Description: Gets a specified item from the server as a 32bit integer scalar/vector. Called after an event is built.

Returns: Non-zero integer on failure.

Arguments	Description
stream_id	The returned unique stream identifier.
item_name	A string containing the name of the item.
dest_buff	A buffer for storing the returned values. May be a vector depending upon the item.
size_of_dest_buff	The size of the aforementioned buffer.
num_of_ret_elements	The number of elements that were returned via the buffer.

```
/* C */
#include "libTM.h"

int32_t TM_Get_Item_I4(uint32_t streamid, char *item_name, int32_t *dest_buff, int32_t
size_of_dest_buff, int32_t *num_of_ret_elements);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_Get_Item_I4(int streamid, String item_name, IntBuffer dest_buff, int
size_of_dest_buff, IntBuffer num_of_ret_elements);
```

```
; IDL
```

```
LONG TM_Get_Item_I4(LONG streamid, STRING item_name, LONG dest_buff, LONG
size_of_dest_buff, LONG num_of_ret_elements)
```

4.1.6. `TM_Get_Item_R4`

Description: Gets a specified item from the server as a 32bit floating point scalar/vector. Called after an event is built.

Returns: Non-zero integer on failure.

Arguments	Description
stream_id	The returned unique stream identifier.
item_name	A string containing the name of the item.
dest_buff	A buffer for storing the returned values. May be a vector depending upon the item.
size_of_dest_buff	The size of the aforementioned buffer.
num_of_ret_elements	The number of elements that were returned via the buffer.

```
/* C */
#include "libTM.h"

int32_t TM_Get_Item_R4(uint32_t streamid, char *item_name, float *dest_buff, int32_t
size_of_dest_buff, int32_t *num_of_ret_elements);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_Get_Item_R4(int streamid, String item_name, FloatBuffer dest_buff, int
size_of_dest_buff, IntBuffer num_of_ret_elements);
```

```
; IDL

LONG TM_Get_Item_R4(LONG streamid, STRING item_name, FLOAT dest_buff, LONG
size_of_dest_buff, LONG num_of_ret_elements)
```

4.1.7. `TM_Get_Item_R8`

Description: Gets a specified item from the server as a 64bit floating point scalar/vector. Called after an event is built.

Returns: Non-zero integer on failure.

Arguments	Description
stream_id	The returned unique stream identifier.
item_name	A string containing the name of the item.
dest_buff	A buffer for storing the returned values. May be a vector depending upon the item.
size_of_dest_buff	The size of the aforementioned buffer.
num_of_ret_elements	The number of elements that were returned via the buffer.

```
/* C */
#include "libTM.h"

int32_t TM_Get_Item_R8(uint32_t streamid, char *item_name, double *dest_buff, int32_t
size_of_dest_buff, int32_t *num_of_ret_elements);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_Get_Item_R8(int streamid, String item_name, DoubleBuffer dest_buff, int
size_of_dest_buff, IntBuffer num_of_ret_elements);
```

```
; IDL

LONG TM_Get_Item_R8(LONG streamid, STRING item_name, DOUBLE dest_buff, LONG
size_of_dest_buff, LONG num_of_ret_elements)
```

4.1.8. `TM_Get_Item_Char`

Description: Gets a specified item from the server as a 8bit character scalar/vector. Called after an event is built. Normally used for retrieving strings from the server.

Returns: Non-zero integer on failure.

Arguments	Description
stream_id	The returned unique stream identifier.
item_name	A string containing the name of the item.
dest_buff	A buffer for storing the returned values.
size_of_dest_buff	The size of the aforementioned buffer.
num_of_ret_elements	The number of characters that were returned via the buffer.

```
/* C */
#include "libTM.h"

int32_t TM_Get_Item_Char(uint32_t streamid, char *item_name, char *dest_buff, int32_t
size_of_dest_buff, int32_t *num_of_ret_chars);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_Get_Item_Char(int streamid, String item_name, CharBuffer dest_buff, int
size_of_dest_buff, IntBuffer num_of_ret_chars);
```

```
; IDL

LONG TM_Get_Item_Char(LONG streamid, STRING item_name, BYTE dest_buff, LONG
size_of_dest_buff, LONG num_of_ret_chars)
```

4.1.9. `TM_Close`

Description: Gracefully shuts the connection down between the server and the client. Always recommended for shutting down connections as it reduces the load on the TMlib server.

Returns: Non-zero integer on failure.

Arguments	Description
stream_id	The returned unique stream identifier.

```
/* C */
#include "libTM.h"

int32_t TM_Close(uint32_t streamid);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_Close(int streamid);
```

```
; IDL

LONG TM_Close(LONG streamid)
```

4.1.10. `TM_Select_Server`

Description: Programmatically and temporarily overrides the value of the server parameter for the currently selected site.

Returns: Non-zero integer on failure.

Arguments	Description
server_name_or_ip	A string containing the fully qualified domain name or IP address of a TMlib server.

```
/* C */
#include "libTM.h"

int32_t TM_Select_Server(char *server_name_or_ip);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_Select_Server(String server_name_or_ip);
```

```
; IDL

LONG TM_Select_Server(STRING server_name_or_ip)
```

4.1.11. `TM_Select_Port`

Description: Programmatically and temporarily overrides the value of the port parameter for the currently selected site.

Returns: Non-zero integer on failure.

Arguments	Description
port	An integer containing the TCP port number the TM-lib server resides on.

```
/* C */
#include "libTM.h"
```

```
int32_t TM_Select_Port(int port);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_Select_Port(int port);
```

```
; IDL

LONG TM_Select_Port(LONG port)
```

4.1.12. `TM_Error_Stack`

Description: Based on the string argument, this function returns a description of the last error that occured.

Note

Server errors take precedence over client errors. Thus, if a client error occurs immediately before a server error in the same function call, the user error will be dropped.

Returns: Non-zero integer on failure.

Arguments	Description
stream_id	The returned unique stream identifier.
description_type	A string containing the name of the error data to retreive. Can be one of "name", "description", "message", "code". Described in the section on error handling below.
dest_buff	A buffer for storing the returned string.
size_of_dest_buff	The size of the aforementioned buffer.
num_of_ret_elements	The number of elements that were returned via the buffer. (string length)

```
/* C */
#include "libTM.h"

int32_t TM_Error_Stack(uint32_t stream_id, char *description_type, char *dest_buff,
int32_t size_of_dest_buff, int32_t *sizeof_ret_results);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_Error_Stack(int stream_id, String description_type, CharBuffer dest_buff, int
size_of_dest_buff, IntBuffer sizeof_ret_results);
```

```
; IDL

LONG TM_Error_Stack(LONG stream_id, STRING description_type, BYTE dest_buff, LONG
size_of_dest_buff, LONG sizeof_ret_results)
```

4.1.13. `TM_Set_Position`

Description: Sets the position in the stream via a time value in UR8 format. A UR8 that is zero means the Beginning of Range (BOR).

Returns: Non-zero integer on failure.

Arguments	Description
stream_id	The returned unique stream identifier.
UR8	A number that contains the UR8.

```
/* C */
#include "libTM.h"

int32_t TM_Set_Position(uint32_t stream_id, double UR8);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_Set_Position(int stream_id, double UR8);
```

```
; IDL

LONG TM_Set_Position(LONG stream_id, DOUBLE UR8)
```

4.1.14. `TM_Get_Position`

Description: Gets the position in the stream as a time value in UR8 format.

Returns: Non-zero integer on failure.

Arguments	Description
stream_id	The returned unique stream identifier.
UR8	A buffer to hold the returned UR8.

```
/* C */
#include "libTM.h"

int32_t TM_Get_Position(uint32_t stream_id, double *UR8);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_Get_Position(int stream_id, DoubleBuffer UR8);
```

```
; IDL

LONG TM_Get_Position(LONG stream_id, DOUBLE UR8)
```

4.2. Time Functions

Described below are all the functions that deal with time, from converting between formats to generating unique filenames. The API description is identical to that used in the item function API.

Note

A UR8 is the number of days since Midnight on Jan 1st, 1982. A UR8 is a floating point number of 64bits.

Note

Midnight on Jan 1st, 1982 is equal to a UR8 of '0.0'.

Note

Unless otherwise specified in the API documentation, all date functions operate in the standard C/POSIX locale, such that dates are represented in the English language.

4.2.1. `TM_UR8_from_String_Absolute`

Description: Convert a string of the exact form "07-Jan-1999:18:00:10.627" to a UR8.

Returns: Non-zero integer on failure.

Arguments	Description
UR8	A buffer to store the result of the conversion.
date_string	A string of the exact form "07-Jan-1999:18:00:10.627".

```
/* C */
#include "libTM.h"

int TM_UR8_from_String_Absolute(double *UR8, char *date_string);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_UR8_from_String_Absolute(DoubleBuffer UR8, String date_string);
```

```
; IDL

LONG TM_UR8_from_String_Absolute(DOUBLE UR8, STRING date_string)
```

4.2.2. `TM_UR8_to_String_Absolute`

Description: Convert a UR8 to a string containing a date of the form "07-Jan-1999:18:00:10.627".

Returns: Non-zero integer on failure.

Arguments	Description
UR8	A variable containing a date in UR8 format.
date_string	A string buffer of at least 24 bytes.

```
/* C */
#include "libTM.h"

int TM_UR8_to_String_Absolute(double UR8, char *date_string);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_UR8_to_String_Absolute(double UR8, CharBuffer date_string);
```

```
; IDL
```

```
LONG TM_UR8_to_String_Absolute(DOUBLE UR8, BYTE date_string)
```

4.2.3. `TM_UR8_from_String_Comparison`

Description: Convert a string of the exact form "1983-09-18 11:42:25.056" to a UR8.

Returns: Non-zero integer on failure.

Arguments	Description
UR8	A buffer to store the result of the conversion.
date_string	A string of the exact form "1983-09-18 11:42:25.056".

```
/* C */
#include "libTM.h"

int TM_UR8_from_String_Comparison(double *UR8, char *date_string);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_UR8_from_String_Comparison(DoubleBuffer UR8, String date_string);
```

```
; IDL

LONG TM_UR8_from_String_Comparison(DOUBLE UR8, STRING date_string)
```

4.2.4. `TM_UR8_to_String_Comparison`

Description: Convert a UR8 to a string containing a date of the exact form "1983-09-18 11:42:25.056".

Returns: Non-zero integer on failure.

Arguments	Description
UR8	A variable containing a UR8.
date_string	A string buffer at least 23 bytes long.

```
/* C */
#include "libTM.h"

int TM_UR8_to_String_Comparison(double UR8, char *date_string);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_UR8_to_String_Comparison(double UR8, CharBuffer date_string);
```

```
; IDL

LONG TM_UR8_to_String_Comparison(DOUBLE UR8, BYTE date_string)
```

4.2.5. `TM_UR8_to_String`

Description: Convert a UR8 to a LOCALIZED (if set) string containing a date of the form "18-Sep-1983, 11:42:25.056".

Returns: Non-zero integer on failure.

Arguments	Description
UR8	A variable containing a UR8.
date_string	A string buffer at least 25 bytes long.

```
/* C */
#include "libTM.h"

int TM_UR8_to_String(double UR8, char *date_string);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_UR8_to_String(double UR8, CharBuffer date_string);
```

```
; IDL

LONG TM_UR8_to_String(DOUBLE UR8, BYTE date_string)
```

4.2.6. `TM_UR8_from_Local_Time`

Description: Get the time from the user's computer and convert it to a UR8.

Warning

This function is highly dependent upon the available time resolution of the host operating system! The precision of the returned UR8 value *will* vary.

Returns: Non-zero integer on failure.

Arguments	Description
UR8	A buffer the hold the returned value.

```
/* C */
#include "libTM.h"

int TM_UR8_from_Local_Time(double *UR8);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_UR8_from_Local_Time(DoubleBuffer UR8);
```

```
; IDL

LONG TM_UR8_from_Local_Time(DOUBLE UR8)
```

4.2.7. `TM_UR8_from_YDOY`

Description: Convert the year, day-of-year, and milliseconds into a 64bit floating point number in UR8 format.

Returns: Non-zero integer on failure.

Arguments	Description
UR8	A buffer to hold the returned UR8.
year	A variable containing the year, including century, e.g. 2006. The value must be >= 1982.
day_of_year	A variable containing the number of days since Jan 1st of the specified year, e.g. 156. The value must be >= to 1 and <= 365 (366 on leap years). Day 1 is defined as Midnight January 1st.
milliseconds	A variable containing the number of milliseconds. The value must be >= 0 and <= 999.

```
/* C */
#include "libTM.h"

int TM_UR8_from_YDOY(double *UR8, int year, int day_of_year, int millseconds);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_UR8_from_YDOY(DoubleBuffer UR8, int year, int day_of_year, int millseconds);
```

```
; IDL

LONG TM_UR8_from_YDOY(DOUBLE UR8, LONG year, LONG day_of_year, LONG millseconds)
```

4.2.8. `TM_UR8_to_YDOYH`

Description: Convert a UR8 to the year, day-of-year, and fractional hours.

Returns: Non-zero integer on failure.

Arguments	Description
UR8	A variable containing a UR8.
year	A buffer to hold the returned year.
day_of_year	A buffer to hold the returned day of the year since January 1st.
hour	A buffer to hold the returned fractional hour.

```
/* C */
#include "libTM.h"

int TM_UR8_to_YDOYH(double UR8, int *year, int *day_of_year, double *hour);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_UR8_to_YDOYH(double UR8, IntBuffer year, IntBuffer day_of_year, DoubleBuffer
hour);
```

```
; IDL

LONG TM_UR8_to_YDOYH(DOUBLE UR8, LONG year, LONG day_of_year, DOUBLE hour)
```

4.2.9. `TM_UR8_to_YDOY`

Description: Convert a UR8 to the year, day-of-year, and milliseconds.

Returns: Non-zero integer on failure.

Arguments	Description
UR8	A variable containing a UR8.
year	A buffer to hold the returned year.
day_of_year	A buffer to hold the returned day of the year since January 1st.
milliseconds	A buffer to hold the returned number of milliseconds.

```
/* C */
#include "libTM.h"

int TM_UR8_to_YDOY(double UR8, int *year, int *day_of_year, int *millseconds
);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_UR8_to_YDOY(double UR8, IntBuffer year, IntBuffer day_of_year, IntBuffer
millseconds);
```

```
; IDL

LONG TM_UR8_to_YDOY(DOUBLE UR8, LONG year, LONG day_of_year, LONG millseconds)
```

4.2.10. `TM_UR8_from_YMD`

Description: Convert the year, month, day-of-month, hour, minute, second, millisecond into a 64bit floating point UR8. The month and day-of-month variables are indexed from 1.

Returns: Non-zero integer on failure.

Arguments	Description
UR8	A buffer to hold the returned UR8.
year	A variable containing the year, including century, e.g. 2006. The value must be > 1982.
month	A variable containing the month. The value must be >= 1 and <= 12.
day_of_month	A variable containing the day of month. the value must be >= 1 and <= 31.
hour	A variable containing the hour. The value must be >= 0 and <= 23.
minute	A variable containing the minute. The value must be >= 0 and <= 59.
second	A variable containing the second. The value must be >= 0 and <= 59.
millisecond	A variable containing the millisecond. The value must be >= 0 and <= 999.

```
/* C */
```

```
#include "libTM.h"

int TM_UR8_to_YDOY(double *UR8, int year, int month, int day_of_month, int hour, int
minute, int second, int millisecond);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_UR8_to_YDOY(DoubleBuffer UR8, int year, int month, int day_of_month, int
hour, int minute, int second, int millisecond);
```

```
; IDL

LONG TM_UR8_to_YDOY(DOUBLE UR8, LONG year, LONG month, LONG day_of_month, LONG hour,
LONG minute, LONG second, LONG millisecond)
```

4.2.11. `TM_UR8_to_YMD`

Description: Convert a UR8 into its component year, month, day-of-month, hour, minute, second, millisecond variables. The month and day-of-month variables are indexed from 1.

Returns: Non-zero integer on failure.

Arguments	Description
UR8	A variable containing a UR8
year	A buffer to hold the returned year in four digit format.
month	A buffer to hold the returned month.
day_of_month	A buffer to hold the returned day of month.
hour	A buffer to hold the returned hour.
minute	A buffer to hold the returned minute.
second	A buffer to hold the returned second.
millisecond	A buffer to hold the returned millisecond.

```
/* C */
#include "libTM.h"

int TM_UR8_to_YDOY(double UR8, int *year, int *month, int *day_of_month, int *hour, int
*minute, int *second, int *millisecond);
```

```
// Java
import java.nio.*;
import edu.umn.space.swaves.TM;

int TM.TM_UR8_to_YDOY(double UR8, IntBuffer year, IntBuffer month, IntBuffer
day_of_month, IntBuffer hour, IntBuffer minute,
IntBuffer second, IntBuffer millisecond);
```

```
; IDL

LONG TM_UR8_to_YDOY(DOUBLE UR8, LONG year, LONG month, LONG day_of_month, LONG hour,
LONG minute, LONG second, LONG millisecond)
```

4.3. Function Return Codes

All the possible local return codes for all the TMlib client functions are listed below with a short description.

Table 4. TMlib Return Codes

Code	Description
0	Success
-1000	TMlib internal error. Please report this error if you see it.
-1001	Null pointer error
-1002	Expected data, but got nothing back from server.
-1005	Method was called that the server did not recognize.
-1009	Stream id not found. TMlib function was called with an invalid stream id.
-1100	Error processing an environment variable. Most likely cause is STEREO_Server set to null.
-1101	TMlib server not found. Likely cause is a DNS error or a misspelled server name.
-1102	No memory left to allocate.
-1200	Error creating socket.
-1201	Error connecting to server.
-1202	Error writing to socket.
-1203	Error reading from socket.
-1204	Socket closed prematurely.
-1205	Server returned incorrect number of bytes.
-1206	Read from socket returned less data than expected.
-1207	Server returned more data than the client's buffer can hold.
-1208	Server returned no data even though data was expected.
-1300	JNI Error: Out of memory.
-1301	JNI Error: JNI Exception
-1302	JNI Error: Internal Error.
-1303	JNI Error: Null pointer.
-1400	Time Conversion Error: String format error. The string passed in does not conform.
-1401	Time Conversion Error: Date out of range error.
-1402	Time Conversion Error: Internal error.
-1403	Time Conversion Error: Null pointer error.
-1404	Time Conversion Error: String overflow. String is probably too long.
Other non-zero codes	Other non-zero error codes are generated by the server. The user should utilize the TM_Error_Stack function to find out more information about the error.

5. Example Applications

This section contains simple programs that demonstrate how to retrieve data from the TMlib server. To that end, these examples have been simplified to show only the basics of some common operations. Full featured applications would also include significant error handling.

The code for all of the examples shown below can be found in the examples directory of TMlib client installation directory.

Note

The `STEREO_Server` environment variable must point to a valid TMlib server for all examples to function properly.

5.1. Getting Data

The following programs connect to the TMlib server and request the CCSDS_APID item from a packet event.

```
/*
* language: C
* filename: apid_c.c
*/
#include <stdio.h>

#include "libTM.h" /* make sure this header is in your compiler's include path */

int main(void)
{
int size, ccsds_apid[1];
uint32_t stream_id;

TM_Select_Domain(&stream_id, "stereo", "stereo_a", "swaves", "packet");
TM_Select_Stream_File(stream_id, "realtime");

/* infinite loop */
for (;;) {
TM_Find_Event(stream_id);
TM_Get_Item_I4(stream_id, "CCSDS_APID", ccsds_apid, 1, &size);

printf("APID: %d\n", ccsds_apid[0]);
}

return 0;
}
```

```
; language: IDL
; filename: apid_idl.pro

pro apid_idl

err = TM_Select_Domain(stream_id, 'stereo', 'stereo_a', 'swaves', 'packet')
err = TM_Select_Stream_File(stream_id, 'realtime')

; infinite loop
while (1 EQ 1) do begin
err = TM_Find_Event(stream_id)
err = TM_Get_Item_I4(stream_id, 'CCSDS_APID', ccsds_apid, 1, size)

print, 'APID: ', ccsds_apid
```

```
endwhile

end
```

Note

When passing data in and out of TMlib, the Java language binding requires the use of the java.nio.* classes such as IntBuffer, CharBuffer, et al.

```
// language: Java
// filename: apid_j.java

import java.nio.*;

import edu.umn.space.swaves.*; /* ensure that tmlib.jar is in your CLASSPATH */


public class apid_j {

public static void main(String[] argv) {
int stream_id;
IntBuffer size = IntBuffer.allocate(1);
IntBuffer ccsds_apid = IntBuffer.allocate(1);
IntBuffer idbuf = IntBuffer.allocate(1);
TM tmlib = new TM();

tmlib.TM_Select_Domain(idbuf, "stereo", "stereo_a", "swaves", "packet");
stream_id = idbuf.get(0);

tmlib.TM_Select_Stream_File(stream_id, "realtime");

// infinite loop
for (;;) {
tmlib.TM_Find_Event(stream_id);
tmlib.TM_Get_Item_I4(stream_id, "CCSDS_APID", ccsds_apid, 1, size);

System.out.println("APID: " + ccsds_apid.get(0));
}
}
}
```

5.2. Handling Errors

All TMlib routines return a signed integer indicating success or failure of the function. If the integer is less than zero, a fatal error occurred and the program should try to determine the cause of the error before continuing. If the integer is greater than zero, then a non-fatal error occurred and the program has the option to ignore the error (but it is still wise to handle it). Of course, if the return value is zero, then the function returned properly without any error.

When an error occurs, whether it be erver side or client side, a non-zero integer is returned as stated above. The program can then call `TM_Error_Stack` to retrieve more information about the error. The possible types of requests to `TM_Error_Stack` are listed in the following table.

Table 5. TM_Error_Stack Requests

"name"	Short title for the error
"description"	Generic description of the error
"message"	Specific description of the error
"code"	Numerical code representing this error

The following examples request a non-existent item from the packet event to generate an error.

```
; language: IDL
; filename: errorstack_demo_idl.pro

pro errorstack_demo_idl
ErrorBuffSize = 2500

err = TM_Select_Domain(stream_id, 'stereo', 'stereo_a', 'swaves', 'packet')
err = TM_Select_Stream_File(stream_id, 'realtime')

err = TM_Find_Event(stream_id)

; There isn't an item named aaaaTTTTxxxQ
err = TM_Get_Item_I4(stream_id, "aaaaTTTTxxxQ", itemBuffer, 1, size)
if (err ne 0) then begin
print, "Error = ", err

err = TM_Error_Stack(stream_id, "name", ErrorDump, ErrorBuffSize, size)
print, "Error Name = ", ErrorDump

err = TM_Error_Stack(stream_id, "description", ErrorDump, ErrorBuffSize, size)
print, "Error Description = ", ErrorDump

err = TM_Error_Stack(stream_id, "message", ErrorDump, ErrorBuffSize, size)
print, "Error Message = ", ErrorDump

err = TM_Error_Stack(stream_id, "code", ErrorDump, ErrorBuffSize, size)
print, "Error Code = ", ErrorDump
endif
end
```

```
/*
* language: C
* filename: errorstack_demo_c.c
*/

#include <stdio.h>

#include "libTM.h"

int main(void)
{
int err, size, itemBuffer[1], ErrorBuffSize = 2500;
uint32_t stream_id;
char ErrorDump[ErrorBuffSize];

TM_Select_Domain(&stream_id, "stereo", "stereo_a", "swaves", "packet");
TM_Select_Stream_File(stream_id, "realtime");

TM_Find_Event(stream_id);

/* There isn't an item named aaaaTTTTxxxQ */
err = TM_Get_Item_I4(stream_id, "aaaaTTTTxxxQ", itemBuffer, 1, &size);
if (err != 0) {
printf("Error = %d\n", err);

TM_Error_Stack(stream_id, "name", ErrorDump, ErrorBuffSize, &size);
printf("Error Name = %s\n", ErrorDump);

TM_Error_Stack(stream_id, "description", ErrorDump, ErrorBuffSize, &size);
printf("Error Description = %s\n", ErrorDump);

TM_Error_Stack(stream_id, "message", ErrorDump, ErrorBuffSize, &size);
printf("Error Message = %s\n", ErrorDump);

TM_Error_Stack(stream_id, "code", ErrorDump, ErrorBuffSize, &size);
printf("Error Code = %s\n", ErrorDump);

}

return 0;
}
```

Note

When passing data in and out of TMlib, the Java language binding requires the use of the java.nio.* classes such as IntBuffer, CharBuffer, et al.

```
// language: Java
// filename: errorstack_demo_j.java

import java.nio.*;

import edu.umn.space.swaves.*;

public class errorstack_demo_j {
public static void main(String argv) {
int err;
int stream_id = 0;
int ErrorBuffSize = 2500;
IntBuffer size = IntBuffer.allocate(1);
IntBuffer itemBuffer = IntBuffer.allocate(1);
IntBuffer idbuf = IntBuffer.allocate(1);
CharBuffer ErrorDump = CharBuffer.allocate(ErrorBuffSize);
TM tmlib = new TM();

tmlib.TM_Select_Domain(idbuf, "stereo", "stereo_a", "swaves", "packet");
stream_id = idbuf.get(0);
tmlib.TM_Select_Stream_File(stream_id, "realtime");

tmlib.TM_Find_Event(stream_id);

// There isn't an item named aaaaTTTTxxxQ
err = tmlib.TM_Get_Item_I4(stream_id, "aaaaTTTTxxxQ", itemBuffer, 1, size);
if (err != 0) {
System.out.println("Error = " + err);

tmlib.TM_Error_Stack(stream_id, "name", ErrorDump, ErrorBuffSize, size);
System.out.println("Error Name: " + ErrorDump.toString());

tmlib.TM_Error_Stack(stream_id, "description", ErrorDump, ErrorBuffSize,
size);
System.out.println("Error Description: " + ErrorDump.toString());

tmlib.TM_Error_Stack(stream_id, "message", ErrorDump, ErrorBuffSize, size);
System.out.println("Error Message: " + ErrorDump.toString());

tmlib.TM_Error_Stack(stream_id, "code", ErrorDump, ErrorBuffSize, size);
System.out.println("Error Code: " + ErrorDump.toString());
}
}
}
```

5.3. Chooser

Below is a chooser based program written in various languages. Notice that the only differences between these and their above counterparts are the arguments to `TM_Select_Stream` and the loop that terminates when there are no more events, i.e. we reached the end of the file.

```
/*
* language: C
* filename: chooser_demo.c
*/
#include <stdio.h>

#include "libTM.h"

int
main(void)
{
int err = 0, size, ccsds_apid;
uint32_t stream_id;

err = TM_Select_Domain(&stream_id, "stereo", "stereo_a", "swaves", "packet");
err = TM_Select_Stream_File(stream_id, "chooser");

/* loop until no more events */
while (err == 0) {
err = TM_Find_Event(stream_id);

TM_Get_Item_I4(stream_id, "CCSDS_APID", &ccsds_apid, 1, &size);

printf("APID: %d\n", ccsds_apid);
}
printf("file read\n");
return 0;
}
```

```
; language: IDL
; filename: chooser_demo.pro

pro chooser_demo

err = TM_Select_Domain(stream_id, "stereo", "stereo_a", "swaves", "packet")
err = TM_Select_Stream_File(stream_id, "chooser")

; loop until there are no more events
while (err eq 0) do begin
err = TM_Find_Event(stream_id)

g = TM_Get_Item_I4(stream_id, "CCSDS_APID", ccsds_apid, 1, size)

print, "APID: ", ccsds_apid
endwhile

print, "File read"
end
```

```
// language: Java
// filename: chooser_demo.java
import java.nio.*;

import edu.umn.space.swaves.*;


public class chooser_demo {
public static void main(String[] args) {
IntBuffer streamid_buf = IntBuffer.allocate(1);
IntBuffer numvals = IntBuffer.allocate(1);
int streamid, err;
IntBuffer ItemI4Buffer = IntBuffer.allocate(1);
TM tmlib = new TM();

tmlib.TM_Select_Domain(streamid_buf, "stereo", "stereo_a", "swaves", "packet");
streamid = streamid_buf.get(0);

err = tmlib.TM_Select_Stream_File(streamid, "chooser");

while (err == 0) {
```

```
err = tmlib.TM_Find_Event(streamid);
tmlib.TM_Get_Item_I4(streamid, "CCSDS_APID", ItemI4Buffer, 1, numvals);
System.out.println("APID: " + ItemI4Buffer.get(0));
}
System.out.println("EOF");

return;
}
}
```

5.4. packets.c

Below is the source to the **packets** application in the C language only. It demonstrates how to get different items, how to convert a time into a string, and how to handle errors when they arise. The source for the other language versions of `packets` is located in the `examples` subdirectory of the TMlib client installation.

```
/**
* This class will call the TM Lib methods and connect to the server and pull
* packet events.  It will then produce an output of the form:
*
* 000001 2004.03.05 21:47:30.3 0d 01 c0 00 01 09 56 db 53 f2 56 00 00 00 ... 00 00
* 000002 2004.03.05 21:47:33.5 0d 04 c0 00 01 09 56 db 53 f5 8b 00 0b 00 ... 00 00
* 000003 2004.03.05 21:47:35.3 0d 01 c0 00 01 09 56 db 53 f7 56 00 00 00 ... 00 00
* 000004 2004.03.05 21:47:37.5 0d 04 c0 00 01 09 56 db 53 f9 8b 00 0b 00 ... 00 00
* 000005 2004.03.05 21:47:40.3 0d 01 c0 00 01 09 56 db 53 fc 56 00 00 00 ... 00 00
* 000006 2004.03.05 21:47:41.5 0d 04 c0 00 01 09 56 db 53 fd 8b 00 0b 00 ... 00 00
* 000007 2004.03.05 21:47:45.3 0d 01 c0 00 01 09 56 db 54 01 56 00 00 00 ... 00 00
* 000008 2004.03.05 21:47:45.5 0d 04 c0 00 01 09 56 db 54 01 8b 00 0b 00 ... 00 00
* 000009 2004.03.05 21:47:49.5 0d 04 c0 00 01 09 56 db 54 05 8b 00 0b 00 ... 00 00
* 000010 2004.03.05 21:47:50.3 0d 01 c0 00 01 09 56 db 54 06 56 00 00 00 ... 00 00
*
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <libTM.h>

#define MAXBUFSIZE 2000

void do_error(uint32_t, int32_t);
void usage(const char *);

int
main(int argc, char *argv[])
{
char mesg[MAXBUFSIZE], stream[MAXBUFSIZE], progname[MAXBUFSIZE], *s;
int j;
int32_t chksum, intbuf[MAXBUFSIZE];
int32_t err, numvals;
uint32_t sid, count = 0;
double doublebuf[MAXBUFSIZE];

strcpy(stream, "realtime");
strcpy(progname, *argv);

while (--argc > 0 && (*++argv)[0] == '-') {
for (s = argv[0]+1; *s != '\0'; s++) {
switch (*s) {
case 'h':
usage(progname);
exit(0);
break;
case 'c':
strcpy(stream, "chooser");
break;
default:
usage(progname);
exit(0);
break;
}
}
}

if ((err = TM_Select_Domain(&sid, "stereo", "stereo_a", "swaves", "packet")) != 0)
do_error(sid, err);

if ((err = TM_Select_Stream(sid, stream)) != 0)
do_error(sid, err);

for (;;) {
if ((err = TM_Find_Event(sid)) != 0)
do_error(sid, err);

if ((err = TM_Get_Item_R8(sid, "CCSDS_SCET_UR8", doublebuf, MAXBUFSIZE,
&numvals)) != 0)
do_error(sid, err);
if ((err = TM_UR8_to_String(doublebuf[0], mesg)) != 0)
do_error(sid, err);

(void) printf("%06d %s ", count, mesg);

if ((err = TM_Get_Item_I4(sid, "CCSDS_HEADER_BYTES", intbuf, MAXBUFSIZE,
&numvals)) != 0)
do_error(sid, err);

for (j = 0; j < numvals; j++)
(void) printf("%02x ", intbuf[j]);

if ((err = TM_Get_Item_I4(sid, "CCSDS_MEAT", intbuf, MAXBUFSIZE, &numvals)) !=
0)
do_error(sid, err);

(void) printf("%02x..%02x %02x ", intbuf[0], intbuf[numvals - 2],
intbuf[numvals - 1]);

if ((err = TM_Get_Item_I4(sid, "CCSDS_ENTIRE_PACKET", intbuf, MAXBUFSIZE,
&numvals)) != 0)
do_error(sid, err);

chksum = 0;
for (j = 0; j < numvals; j++)
chksum += intbuf[j];

(void) printf("(%02x)\n", chksum & 0xFF);
(void) fflush(stdout);
/* roll the counter over */
count == 999999 ? count = 0 : count++;
}
TM_Close(sid);
return (0);
}

void
do_error(uint32_t sid, int32_t err)
{
char mesg[MAXBUFSIZE];
int32_t numvals;
int isc;

if (err == 0)
return;
```

```
if ((isc = TM_Error_Stack(sid, "name", mesg, MAXBUFSIZE, &numvals)) == 0)
(void) fprintf(stderr, "Error Name: %s\n", mesg);
else
(void) fprintf(stderr, "Fatal error. Cound not get Error Stack.\n");
if ((isc = TM_Error_Stack(sid, "message", mesg, MAXBUFSIZE, &numvals)) == 0)
(void) fprintf(stderr, "Error Message: %s\n", mesg);
else
(void) fprintf(stderr, "Fatal error. Cound not get Error Stack.\n");
if ((isc = TM_Error_Stack(sid, "description", mesg, MAXBUFSIZE, &numvals)) == 0)
(void) fprintf(stderr, "Error Description: %s\n", mesg);
else
(void) fprintf(stderr, "Fatal error. Cound not get Error Stack.\n");
if ((isc = TM_Error_Stack(sid, "code", mesg, MAXBUFSIZE, &numvals)) == 0)
(void) fprintf(stderr, "Error Code: %s\n", mesg);
else
(void) fprintf(stderr, "Fatal error. Cound not get Error Stack.\n");
(void) fflush(stderr);

if (err < 0)
exit(-1);
}

void
usage(const char *progname)
{
fprintf(stderr, "usage: %s [-c] [-h]\n", progname);
fprintf(stderr, "-h\t\tThis message.\n");
fprintf(stderr, "-c\t\tUse chooser to select stream.\n");
}
```

5.5. Building and Running the Example Programs

This section describes how to compile, run, and execute the example programs included in this distribution of the TMlib client. For the C language some familiarity with **make** and **gcc** are recommended, but not strictly necessary for the examples on Mac OS X and other Unix-like platforms. Similarly, some familiarity with the Java toolchain is also recommended, but again, not required. The IDL examples are by far the easiest to get running if the TMlib client installation is correct, since they are the most straightforward to compile and run.

When compiling the examples for C on Mac OS X, be aware the the C library has dependencies upon libcurses and libxml2. This means that your linker should be able to find these libraries to link against, otherwise you will run into errors. Similarly, under Windows, `TMlib.DLL` has dependencies upon a few DLLs (`libxml.DLL`, `libz.DLL`, and `libiconv.DLL`) which are included in the Windows TMlib distribution. As long as you link against the `TMlib.LIB` linker file and the other libraries are in your `PATH` (the path to them is inserted in the `PATH` environment variable at installation automatically) you should have no problem.

5.5.1. C

Building the C programs on Mac OS X is fairly trivial using the included makefile. Change directories to the `examples/c` directory in the TMlib client installation tree. As long as you have **make** and **gcc** installed and they are in your `PATH`, the following command will build all the examples.

```
unix% pwd
/Applications/STEREO/TMlib_Client/examples/c
unix% make -f makefile.example
```

The **pwd** command verifies you are in the correct directory. If the compilation is successful, you will end up with a set of binaries in your current directory.

Note

Compiling the C examples under Windows is development environment dependent. To link with the TMlib library, included in the Windows distribution is the linker file TMlib.LIB which is located in `C:\Program Files\STEREO\TMlib_Client\lib\c`. See the above discussion regarding TMlib's dependency upon other libraries.

5.5.2. Java

The Java examples are a little bit easier to compile. As long as you have the `tmlib.jar` file in your `CLASSPATH` and you've verified that your TMlib installation works correctly (see section 3 of this document), then the following command will compile all of the Java examples on Mac OS X.

Note

If TMlib was installed in the default location, the path to the JAR file containing the methods will be `/Applications/STEREO/TMlib_Client/lib/java/tmlib.jar` on Mac OS X and `C:\Program Files\STEREO\TMlib_Client\lib\java\tmlib.jar` on Windows XP

```
unix% pwd
/Applications/STEREO/TMlib_Client/examples/java
unix% javac *.java
```

The only difference to building the Java examples on Windows is that your current directory will be `C:\Program Files\STEREO\TMlib_Client\examples\java` instead of what is listed above for the Mac OS X example.

5.5.3. IDL

As mentioned above, IDL is the easiest to get the examples up and running in. Again, verify that your environment is setup properly and start the IDL environment with the **sidl** shell alias command in the directory containing the IDL examples. After starting the IDL environment, just compile the example and run it as shown below in thie Mac OS X example.

```
[01:23 PM] hobbes$ pwd
/Applications/STEREO/TMlib_Client/examples/idl
[01:23 PM] hobbes$ sidl
IDL Version 6.1.1, Mac OS X (darwin ppc m32). (c) 2004, Research Systems, Inc.
% Compiled module: TM_GETLIBPATH.
% Compiled module: TM_SELECT_SERVER.
% Compiled module: TM_SELECT_PORT.
% Compiled module: TM_SELECT_DOMAIN.
% Compiled module: TM_SELECT_STREAM.
% Compiled module: TM_FIND_EVENT.
% Compiled module: TM_GET_ITEM_I4.
% Compiled module: TM_GET_ITEM_R4.
% Compiled module: TM_GET_ITEM_R8.
% Compiled module: TM_GET_ITEM_CHAR.
% Compiled module: TM_ERROR_STACK.
% Compiled module: TM_HELP.
% Compiled module: TM_GET_POSITION.
% Compiled module: TM_SET_POSITION.
% Compiled module: TM_CLOSE.
% Compiled module: TM_UR8_TO_YDOY.
% Compiled module: TM_UR8_TO_YMD.
% Compiled module: TM_UR8_TO_STRING.
% Compiled module: TM_UR8_TO_STRING_FR.
% Compiled module: TM_UR8_TO_STRING_ABSOLUTE.
% Compiled module: TM_UR8_TO_STRING_COMPARISON.
% Compiled module: TM_UR8_FROM_YDOY.
% Compiled module: TM_UR8_FROM_YMD.
% Compiled module: TM_UR8_FROM_STRING_ABSOLUTE.
% Compiled module: TM_UR8_FROM_STRING_COMPARISON.
% Compiled module: TM_UR8_FROM_LOCAL_TIME.
IDL> .r apid_idl
% Compiled module: APID_IDL.
IDL> apid_idl
APID:         1285
APID:         1285
APID:         1298
APID:         1287
APID:         1298
...snip...
```

Note

If for some reason you wish to compile the TMlib routines yourself or the **sidl** command does not work, just start IDL as you would normally, and since the `IDL_PATH` variable points to the `TMlib.pro` file, all you have to do is type **.r TMlib.pro** and everything will work.

On the Windows platform, just open the IDLDE and the TMlib routines will automatically be compiled and available for you thanks to the `IDL_STARTUP` environment variable. Next, navigate to the `example/idl` directory of your TMlib installation and open the `apid_idl.pro` file. Then just click "compile" and "run" in the IDLDE and the example should start outputting the same information as the Mac OS X example above.

Note

If for some reason you wish to compile the TMlib routines yourself or the IDL_STARTUP environment variable does not work, just start IDL as you would normally, and since the `IDL_PATH` variable points to the `TMlib.pro` file, all you have to do is type **.r TMlib.pro** and everything will work.

5.5.4. Fortran (Mac OS X only)

Coming Soon!

6. Tips, Tricks, and Notes

Included below are a few random notes, tips, tricks, and troubleshooting suggestions worth reading over.

Note

On Unix-like platforms each language binding is distributed as its own library due to library format constraints. On Windows, all language bindings can and are distributed as one DLL.

Note

As stated above, setting `STEREO_TMlib_Client_Home` to the proper value is critical for the proper operation of TMlib. TMlib uses this environment variable to search for many things including the XML error database file as well as the native library that Java's JNI or IDL's CALL_EXTERNAL use. This variable is set automatically during installation and should only be changed if the TMlib client installation is moved for some reason.

Note

On Windows the `PATH` variable is just as crucial to the proper operation of TMlib as the `STEREO_TMlib_Client_Home` variable is. Windows uses the `PATH` variable to find the location of the library when applications ask to use it. Again, this variable is set by the installer during installation and should not be changed unless the installation itself has moved.

Tip

If you having trouble getting any of the applications to run, look first at the environment variables listed in the previous seections and double check to make sure they are set correctly. For example, the `STEREO_TMlib_Client_Home` environment variable should be set to the base directory of the TMlib installation. For a vanilla installation on Mac OS X, this would be the value `/Applications/STEREO/TMlib_Client`. On Windows, the value of this variable would be `C:\Program Files\STEREO\TMlib_Client`

Tip

Always make sure that your `CLASSPATH` is set properly to be able to find `tmlib.jar`. This JAR file is automatically added to your `CLASSPATH` upon installation on both Windows and Mac OS X.