GTOSS

"Getting Started"

GTOSS Version H12.1

Release for Ubuntu Distribution of Linux

(likely compatible with all Linux and Unix variants)

Oct 2008

Developed By
David D. Lang Associates
Seattle, WA

Originally for
NASA Johnson & Marshall Space Flight Centers
Then On-going for
Various Industry Advocates & Supporters

ACKNOWLEDGEMENT

The author of GTOSS would like to acknowledge Keith Curtis and Brad Edwards for their encouragement and support in making this release of GTOSS available to the Space Elevator Community.

Table of Contents

OVERVIEW – What is GTOSS

GTOSS stands for **G**eneralized **T**ethered **O**bject **S**imulation **S**ystem. It has a rich genealogy dating back almost 30 years to its initial development under the sponsorship of NASA; in the intervening time, it has been used by 30 organizations, including government organizations and large aerospace companies. More can be found about the evolutionary details of GTOSS on the web site: http://home.comcast.net/~GTOSS/.

The current GTOSS software system consists of a large body of Fortran 95 source-code and 10 reference manuals describing all phases of user operation, mathematical model derivations, and system software design. GTOSS programming is characterized by top down design, object oriented structure, modular isolation of environment models, and convenient software *hooks* provided for user-specific modifications. The code is constrained to a *highly portable* subset of Fortran 95, and can run on virtually all computers (with more or less minor modifications required to adhere to system/compiler idiosyncrasies). The ability to conduct automated verification of GTOSS execution is provided to substantiate correct installation and/or user code modifications if desired. This particular incarnation of GTOSS is specifically aimed at a Linux/Unix installation; in particular, it should be a "turn-key" installation for the Ubuntu distribution of Linux.

It must be stated up front that GTOSS is complex, which, in a sense, simply reflects the complexity of the subject matter itself; namely, GTOSS allows the simulation of an arbitrary number of bodies, connected in arbitrary fashion, by an arbitrary number of tethers. Thus, it could simulate flight of a sailing Frigate's "chained-cannonball" shot; or, a wind-tossed dirigible constrained by multiple mooring lines; or, a kite; or, the electro-dynamic behavior of connected bare-wire tethers in orbit; or, the modern day space elevator; *and everything in between*. Dynamics is a complicated scientific/engineering discipline, and tether dynamics is doubly so. Correct results to tether dynamics problems are often non-intuitive, and complex to interpret.

As you use GTOSS, before you conclude it is in error, it should be pointed out that over a period of 30 years of serving GTOSS users of all levels and disciplines, that 99 % of the problems that were encountered by users have proven to be of the users' own making due to having not read the documentation, or a general lack of dynamics/engineering experience; that said, legitimate bug-detections/reports are always welcome ☺.

The GTOSS Applications

The GTOSS complex consists of 5 executable programs:

GTOSS – Conducts the actual tethered object simulation run
DTOSS – A "post-processor" to provide **printer-friendly** data output
CTOSS – A "post-processor" to provide **graphing-program-friendly** data output
UTOSS – A "post-processor" to provide other useful **utility-data output**
VTOSS – Execute to accomplish the **automated run-verification** capability

The next section elaborates on the actual rolls played by the above application programs:

How GTOSS Works

The GTOSS simulation program,

(1) **First accepts an input text file created by the user per a format specified in the documentation (said text file defining the dynamical system to be examined)**;

(2) **Then performs a time-domain simulation of the specified dynamic situation.**

(3) **As this simulation unfolds in time, GTOSS periodically dumps data describing all aspects of this simulation into a set of "Results Data Base" files (called the RDB); this data is essentially an exhaustive compendium of analytical variables pertaining to a GTOSS solution.**

As the last step, the Post Processor Programs come into use;

(4) **After the simulation has completed, the user can then run Post Processing programs against the GTOSS-generated RDB to produce user-friendly results.**

Note: These post-processed results above could be text "printer-friendly formats", "column-delimited formats" (friendly to graphic-display programs), "data formatted to drive specific animation programs" (for example, data complying with an API defined per an animation application), or even, "arbitrarily formatted data" as defined by the user, who would create a sub-program that would access the RDB (via an intermediary GTOSS program called UTOSS, designed with just this intent in mind).

A Note on the RDB Scheme: This scheme exists because one frequently never knows *in advance* just what data ultimately are going to be of interest or need examining after a simulation has completed. Thus, were the desired output variables necessarily specified before a run, to be dumped during the run, and if after the run, the user wished other variables could be examined, the run would have to be repeated, clearly a significant waste of lapsed-time as well as CPU resources. Since complex tether simulations often take hours (and even days) to complete, it represents a significant use of resources to re-do a run. The RDB, by capturing all the data from a run becomes then a way to preserve this resource investment. The definition of this data can be related to the analytical variables defined in the GTOSS equations document. More information can be found about this in the RDB (ie. RTOSS) User manual.

The POST PROCESSORS (PP's)

All these programs that provide access to the RDB in behalf of the user are called "Post processors" (PP's). There is a large number of user-friendly options already available within each of these PP's. The PP output options typically are data synopses that are applicable to aspects of specific types simulation explorations, and other useful general

aggregations of data pertaining to dynamics interpretations. The definitions of these options are found in the respective PP user manuals (as well as the Quick Ref manual).

The specific Post Processor Programs provided are:

DTOSS: generates a text-file layout format (friendly to user visual perusal)
CTOSS: generates column-delimited text output (friendly to plotting programs)
UTOSS: provided to allow a user flexibility beyond that available in either DTOSS or CTOSS for post processing of the RDB. For instance, this PP is where various sub-programs reside that create output formats specifically defined to drive animation programs; this is also where the output data format is created to accomplish automated verification of GTOSS. This PP is equally aimed at supporting outputs that are binary as well as text oriented.

The above three PP's can be thought of as an *intermediary* standing in as a tool for the user to gain access to the "analytically defined variables" of GTOSS that reside in the RDB. The definition of the RDB format is generally of no concern to a GTOSS applications user; only the set of subroutines constituting the RDB subsystem (called RTOSS) need be privy to details of the RDB design. Thus during runtime, the RTOSS routines grab the analytical variables of GTOSS dynamics definition, and stow them in the RDB; then, at post processing time, a set of RTOSS co-routines reconstitute the time histories of these same analytical variables for the benefit of the user display. The PP's then aggregate these analytical variables in user-friendly arrangements for output.

Note: If a user wants to expand on a particular PP's option offerings, this is easily done via cloning then modifying a provided "stubb" and declaring it as a new output option for that PP (as opposed to writing a whole new *standalone* PP program to access the RDB); adding a clone can often be done in 10 or 15 minutes for those who are familiar with this. For instance, suppose a user wants to create a column-delimited data file that contains a *particularly* convenient subset of data; the user would either clone and modify an existing output option that was similar to what was desired, or clone and flesh-out a "stubb". In all cases, the user would presume upon the PP to actually take care of details of accessing the analytical variables of the RDB, rather than attempting to deal directly with the schema used by GTOSS to stash this data in the RDB.

VTOSS: This is not a PP in the above sense, however, it is a program that can be run to compare data that has been previously output (in a special validation-data format) by the post processor UTOSS in such a way as to allow validation (ie. comparison between a "Reference set of data" (usually generated from a previously verified version of GTOSS), and the identical data items as generated by the new version of GTOSS that is a candidate for verification.

User Defined Input to GTOSS

When GTOSS wakes up during execution, it first looks around the directory (or folder) in which it launched, for a file with the explicit name "**INGOSS**" (note, this must be upper-case, as "case-counts" in Unix); GTOSS views this file as containing all the information needed to define the dynamical system and thus commence executing the simulation. Failing to find an INGOSS file in its local directory, GTOSS next looks for a file named "**RTPATH**"; this file would then be assumed to provide information about where GTOSS can find the intended INGOSS file (thus, RTPATH is a directory "re-direction" file). Under all conditions, failure to identify and open a requested "file name" will precipitate a Fortran file-open-failure system warning message and a run-time abort.

The format and contents of the INGOSS file is explicitly specified within the GTOSS documentation. Furthermore, a "*skeleton version*" of this file is included within the GTOSS delivery, found here: "**A__RUN / GRUNS / _GSKL_vH12**". This "GSKL" file is rather voluminous as it contains EVERY system configuration data item, as well as execution options offered by ALL features of GTOSS, thus it can be daunting at first sight. A good way to deal with this complexity is to build-on existing INGOSS files that have already been setup to define typical situations with appropriate execution option features.

For instance, within the main delivery-directory (folder) there can be found a set of 36 INGOSS files located at: "**A__RUN / GRUNS / GRUNOO …thru… GRUN35**. These correspond to a myriad of tethered dynamics situations. These files are included as part of the exhaustive array of executions that GTOSS must perform to pass auto-validation, BUT, in addition, they also represent a wealth of examples that can be used as starting points for building new situation INGOSS files. These runs are generally easier to deal with because they have been stripped of many of the options that are extraneous to a particular application, and feature a narrowed sub-set of geometry configuration data.

Of course, one can always start with the full-content GSKL file and systematically delete non-applicable items, however even for a knowledgeable GTOSS user this can become tedious, thus, said users usually resort to this approach somewhat infrequently.

The post processors behave in a totally parallel fashion to GTOSS regarding their attempt to read input. They too look for a primary input file, and failing to find same, then turn to a possible *alternate* input path re-direction, namely, that same file RTPATH that GTOSS would use (thus RTPATH defines input re-direction to ALL GTOSS application). The only difference is in the name of the input files that each program ultimately reads; these designations are shown below along with their locations in the delivery file set:

DTOSS – reads IN**D**OSS (see "A__RUN / DRUNS / _DSKLv04/ DRUN00…DRUN35)

CTOSS – reads IN**C**OSS (see "A__RUN / CRUNS / _CSKLv04/ CRUN04…misc selec)

UTOSS – reads IN**U**OSS (see "A__RUN / URUNS / _USKLv03/ URUN00…URUN35)

VTOSS – reads IN**V**OSS (see "A__RUN / VRUNS / _VSKLv02/ VAFOWF1…misc)

Delivery Files and Suggested Directory Structure

The software delivery consists of five distinct elements:

1. This "**Getting Started**" document,
2. Directory containing **GTOSS User Reference Document set** (10 pdf files),
3. Directory containing **"getting started" first example GTOSS run for users**
4. Directory containing Extensive **GTOSS Space Elevator Dynamics Explorations**
5. Directory containing:
   Source Code Directories (and sub-directories),
   Unix/Linux Scripts for "make-files", and other useful operations
   Input Text File Directories (for reference and GTOSS verification)

NOTE: *The source-code delivery structure is chosen specifically so as to expedite the creation of GTOSS using delivered make-files; the hierarchical directory order of this delivery is critical because source-code references to include-files use path-names that assume this specific source-code directory structure; so to compile as a "turn-key operation"(with no need to touch source code), requires the suggested source-code structure below*.

So, one would start by creating a directory, say, **/myGTOSS/** and deposit delivery conten into that directory; this would result in a directory structure containing "at a minimum" the files and directories shown below:

/myGTOSS/ ← the directory you create to contain all the delivery files

- **/A_HDR/** ← source for GTOSS include files & F95 modules & data classes
- **/A_GOSS/** ← source for main program host for GTOSS
- **/A_TOSS/** ← source for GTOSS sub-system for TOSS Bodies
- **/A_FOSS/** ← source for GTOSS sub-system for Tethers
- **/A_ROSS/** ← source for GTOSS RDB sub-system
- **/A_BOSS/** ← source for GTOSS Flexible boom RDB sub-system
- **/A_ENVR/** ← source for natural environment models
- **/A_UTIL/** ← source for general Utility routines

- **/A_DOSS/** ← source for Post Processor program DTOSS
- **/A_COSS/** ← source for Post Processor program CTOSS
- **/A_UOSS/** ← source for Post Processor program UTOSS
- **/A_VOSS/** ← source for Post Processor program VTOSS

- **/A__RUN/** ← input data files for reference and for verification
- **/verify GTOSSv12/** ← data images of correct reference version of GTOSS
- …the delivery script files… ← GNU make-files and other useful scripts

Revealed below are the contents of various chosen directories from above:

/A__RUN/

/GRUNS/ ← INGOSS files for verification and reference
/DRUNS/ ← INDOSS files for verification and reference
/URUNS/ ← INUOSS files for verification and reference
/CRUNS/ ← INCOSS files for chosen runs defined in /GRUNS/
/VRUNS/ ← INVOSS files for chosen examples
Nom SE Ribbon Data ← Text Data snippet for specifying the nominal tapered Space Elevator ribbon specification within an IGOSS file to reflect the original ribbon design of Brad Edwards.
/INGOSS_Late_Start_run_setup/ ← Examples of specifying the "Late Start" feature

….assorted script files….. ← GNU make-files and other scripts shown as below:
MakeGTOSS ← GNU make-file to build GTOSS
MakeDTOSS ← GNU make-file to build DTOSS
MakeCTOSS ← GNU make-file to build CTOSS
MakeUTOSS ← GNU make-file to build UTOSS
MakeVTOSS ← GNU make-file to build VTOSS
MakeALL ← Linux Script file to build all the GTOSS programs
ALL_cleanwell ← Linux Script file to delete all GTOSS ".o" files
RTPATH ← Re-direction file for GTOSS inputs (optional usage only!)
/case-convert script/ ← Contains script to convert Dir/File names->upper case (if needed)

/verify GTOSSv12/

INVOSS ← Input to VTOSS (INVOSS) used to verify GTOSSvH12.1
/OUTDISvH121/ ← Contains Output data files from DTOSS for the 36 ref. runs
/VERIFYvH121/ ← Contains output from UTOSS for GTOSS verification w/VTOSS

/SE_ Lib_Run_Example/

INGOSS (SE Lib) ← GTOSS Sample Run input file
INCOSS (SE Lib) ← CTOSS Post Processor input file

/Your First GTOSS run/

INGOSS (SE Lib) ← GTOSS input file to create a Space Elevator example run
INCOSS (SE Lib) ← CTOSS Post processing input to create graph-able results

/SE Dynamics Exploration Runs/

Directories containing definitions of 35 different SE dynamics explorations!

Installing GTOSS

Due to the idiosyncrasies of various computer platforms and operating systems, the best way to acquire GTOSS is to simply compile an executable version of the program from the Source code itself.

It is assumed that the user is at least functionally comfortable with Unix or Linux. While this delivery was developed under **Ubuntu**, a particular Linux distribution, it should be fairly applicable to all such variants of Unix (for example, the Darwin Unix variant underlying the Mac OS X implementation).

The Target Platform

This delivery was specifically developed in the context of Ubuntu running on an Intel PC machine (and arrived thereto from a previous implementation on the Darwin Unix kernel running on a PowerPC Mac). While, all that is really essential to setting up GTOSS is access to a Fortran 95 compiler on your platform, cross-platform/cross-compiler porting can present a multitude of small (but surmountable) nuisances and roadblocks; for bringing up GTOSS under other Ubuntu platforms, such should be virtually non-existent.

A word of warning

Some Linux/Unix Platforms may take liberties with the "text-case" of directory and file names, during the process of porting and reconstitution of source code file structures! In particular, the OS may set all UPPER-case names to Lower-case; this perversity can go deep into the directory structure. This, of course, can wreak havoc upon the Unix make-file/Fortran compiling and directory access protocols! Since it does not touch text "within a file", one can consult the make-files to gain an understanding of how the "naming convention should look". The GTOSS delivery files have bee zipped within a single delivery folder as a means of hopefully avoiding this.

Installing the Fortran Compiler

This delivery was compiled under the GNU Open Source Fortran called g95; another variant of this Fortran, called gFortran, is also available as free-ware. The g95 compiler was found to present no bugs in running through the 60,000 + lines of code that exercised a great many of the advanced features of the Fortran 95 language specifications (complex data classes, pointers, etc).

To get started, visit the g95 web site (or go through GNU directly), and locate the g95 delivery downloads that pertain to your specific OS and CPU platform type. The installation of g95 can be somewhat problematic if you fail to install ALL the support packages needed (or if Ubuntu is missing tools that g95 needs, but were not installed on your original Ubuntu implementation). If this is the case, you will likely not know it until

you attempt your first "executable build" (see next section). The g95 download site strongly recommends utilizing a package installer such as "apt-get" or "Synaptic Package Manager", as opposed to attempting to figure out what is needed and performing a manual install.

Confirm compiler operation by issuing the command "g95" in a Unix "terminal shell". If the compiler has been properly installed and its location (ie. search paths to it) has been made known to the system, then this command should elicit a response from the compiler, identifying itself, and possibly complaining to the effect that the compiler "was invoked but no source file was identified (or some similar diagnostic type response)". On the other hand, if the Terminal shell responds by telling you something like "file or directory not found", then likely what has happened is that your "Unix PATH variable" (which specifies the search routes you proclaim to the system that you want followed in any attempt to honor an application execution attempt) does not include a pathway to g95. To remedy this, then you will want to modify your system "PATH" variable to include a path that leads to the compiler (wherever you may find it to be).

"C" Pre-Processing

To compile properly under Fortran 95, GTOSS requires "C" preprocessing of the source prior to its being submitted to the Fortran compiler. This is a compiler-invoked function that is conventionally triggered by use of the source file name suffix of ".F90". The g95 compiler installation seems to automatically provide such capability if the compiler has been installed properly. Failure to have pre-processing on the GTOSS source will precipitate a myriad of compiler error messages, most likely pertaining to un-recognized or un-declared variable names, Fortran 95 "Code Modules" having compile errors, and missing Modules, etc, etc. It will be evident that something is gravely wrong with the compile process (for instance, it will not silently sneak up on the code so as to render an executable, but dysfunctional GTOSS application).

Building GTOSS

1. At this point it is assumed that the user has created a directory (folder), called, say, **/myGTOSS/** and contained directly-under this directory are all the intact sub-directories of the GTOSS code delivery (see section above on "Delivery Files" for what the source contents of this directory should look like).

2. The various Unix Script files should be removed from their sub-directory, and now also reside as "individual files" directly-under the same user directory **/myGTOSS/**.

3. Open a Unix/Linux "terminal shell", and navigate until the shell's "current working directory" is in fact **/myGTOSS /**. Confirm this by using the "pwd" command.

4. Now attempt a GTOSS compile and link by invoking the GTOSS make-file using the command: "make –f MakeGTOSS".

5. This should start a compilation, as witnessed by a march of lines, each containing: the list of compiler options being used, the GTOSS sub-routine name being compiled (such a "GTOSS", "GTOSUB", etc,), etc.

If you have arrived at this point, then just let the process proceed until completion.

However, it is possible that the response you get to invoking the make-file may be a system diagnostic such as "no such directory or file found". If this is the case, then (barring an error in navigating to **/myGTOSS/**) the make-file may not be recognized as a legitimate Script input. This could be a "permissions type of problem", that can be remedied by executing the following command from the terminal: "chmod 755 MakeGTOSS", which will set proper permission for the Script file to be recognized in that role (this problem can occur with ANY of the Script files contained in this delivery).

Note, at a minimum, the make files must reflect the choice of Fortran compiler. This is specified is the 2nd line in each make-file, and should either be:

FC=g95

or,

FC=gfortran

or,

FC= "your system recognized name of whatever compiler you are using"

Now assume that compilation goes to completion, but a "link error" occurs that interrupts creation of an executable application program. This is usually because a program reference has been made to an external procedure that cannot be found. If the bogus reference is traced to a source code routine as the culprit, then that source module is usually identified, and the problem can be remedied with dispatch! However, if no source procedure is identified, then it is usually a system-level library module that cannot be found (such as "crt1", or some such). In this case, culprit may be the absence of a required (but uninstalled) system library module. Such missing modules can sometimes be attributed to "non-thorough manual-installs of software". This could be true of your Fortran install (and is why g95 recommends using an "installer package"; the g95/GNU website names a couple of candidate package-installers for this). An appropriate Internet search of identifying names of the missing "library module" will usually turn up information leading to the identity of the missing library and information about how to download and install same (you are likely not the first to have encountered this failure).

Building Post Processors

In a manner completely parallel to the GTOSS build procedure described above, the rest of the GTOSS Post Processor executable programs can be built. Instead of using "MakeGTOSS", they would employ

```
make –f  MakeDTOSS
make –f  MakeCTOSS
make –f  MakeUTOSS
make –f  MakeVTOSS
```

Resulting Application Programs

A successful "make" of the GTOSS application programs will yield the following executable files:

```
GTOSSvH12
DTOSSvH12
CTOSSvH12
UTOSSvH12
VTOSSvH12
```

These will all reside in your directory called **/myGTOSS/.**

Make-file Options, and Using the Other Script Files

The first thing to note about all the delivered make-files is the "cleanwell" feature. For instance, many times a change is made in a header (ie. include) file that will affect ALL (or most) of the other sub-routines of GTOSS, etc. In this case, re-running a make-file build will not precipitate a re-compile of all of the subroutines (which, say, is necessary for the include-file change to be properly felt through all code). This is due to the way in which the GNU make-utility manages the selection of which files need to be re-compiled at any particular time; that rule being, if a source file has changed, then re-compile it. The problem is that this decision is triggered only by explicit-changes to a source file, and is oblivious to changes in header files. Thus the user MUST force a full recompile. This can be done by using an argument in the make-file that will simply delete the associated ".o" files related to the make. So, simply execute:

```
make –f  MakeGTOSS cleanwell
```

There will be no terminal response, this will not precipitate a compile, BUT, you will find that the ".o" files will now be gone. Next, then just re-execute the make command, except with the "cleanwell" argument deleted. This will force re-compilation of all the related modules. This same modus operandi will apply to ALL the Post Processor make-files.

If ALL the ".o" files need to be deleted, and a completely clean new build of GTOSS and all the PP's need to be made, use the Scripts: "ALL_cleanwell", followed by "MakeALL". These would be invoked from the terminal via the commands:

./ALL_cleanwell

followed by

./MakeALL

These make-files adhere closely to the standard GNU make-utility definitions. One can fairly well determine what is going on in the make-files by examining their contents, however, before modifying them, make sure to save a copy, as it is easy to enter a place where nothing seems to work properly and what's more there is little to go on as to what is wrong when you enter this "wonderful world of the make-file madness"

Running GTOSS

How To Do It, and What Happens

The preferred way to run GTOSS is to place both a copy of the GTOSS executable AND the INGOSS file in the same directory (Folder) and execute GTOSS there (you can just clone a copy GTOSS to this directory containing the run ingredients). After execution, this folder then will contain the following (note, you start by putting into the directory the "green-files", and after execution the "red-files" will have shown up in the directory):

- **GTOSS** *which you put there*
- **INGOSS** *which you put there*, and serves as a run's concise definition for later ref.
- **RDB files**, *created by GTOSS*, and which can be Post Processed for results display
- **GERROR**, *created by GTOSS*, a text file with info about any errors that occurred
- **OUTQUIK** *created by GTOSS*, a text file time-history synopsis that is written as the simulation unfolds to provide the user a quick-look at how the simulation progressed (it can also be examined during execution also); there are a number of different pageformat definitions available for this synopsis file, pertaining to various generic usages of GTOSS; page selection is made via a parameter in the INGOSS file.

Note, output to a Unix "terminal shell" can also be specified as a means of monitoring execution throughout the GTOSS run (here, the user can again select from an array of terminal output pre-formatted pages).

Thus, expanding this example, an actual GTOSS production-run directory that might result from invoking all the post processors against a GTOSS simulation might look something like this:

- **GTOSS, INGOSS, GERROR, OUTQUIK,**
- **DTOSS, INDOSS, DERROR**
- **CTOSS, INCOSS, CERROR**
- **UTOSS, INUOSS, UERROR**

- **RDB files**
- **The results output file called "OUTDIS",** created by DTOSS
- **The results output file(s)** created by CTOSS
- **The results output file(s)** created by UTOSS

Note: VTOSS is not included above since it is only brought into play to verify new versions of GTOSS code, and is not a part of routine production usage of GTOSS.

"Terminal" Output and other Useful Results

Some Unix/Linux system implementations conveniently do two things automatically for you when you double-click an executable application within a directory (folder).

1. It opens a "terminal shell" with the "current working directory" set to that directory from which the application is started, AND,
2. It starts the application executing.

Thus, if the application is configured to output results to a shell terminal, it can conveniently be monitored as the run proceeds.

This can of course be accomplished explicitly via a manual route, namely, after one has placed the copy of GTOSS in a folder with its associated INGOSS file, then,

1. The user can open a terminal, and set its current working directory to the subject folder, AND,
2. Enters a command to execute GTOSS directly.

Note 1: In the first example, these steps are accomplished automatically for you by the simple act of double-clicking the executable GTOSS; the 2nd scheme is much less convenient, but works fine to accomplish the end result.

Note 2: GTOSS is executed in the later case from the terminal by entering the command "./GTOSS" (or whatever the name of the GTOSS application happens to be).

The GTOSS user can specify Terminal output formats from a selection of output options; these options are noted-in and specified-via a data item in the INGOSS file (also discussed in the Official documentation). In addition, during a run, GTOSS writes to the OUTQUIK file (discussed previously in this document).

Error Reports

Two kinds of errors can appear; System/Fortran runtime errors, and GTOSS perceived errors. Some GTOSS errors are reported via the Terminal, while others are reported via the GERROR file that always attends a GTOSS execution. If any kind of problem

develops, it is wise to check the GERROR file (for the case of Post Processing, the DERROR, CERROR file, etc).

A large number of GTOSS subroutines can encounter and will report problems and then execute a Fortran STOP command; such Stop commands are almost always accompanied by the identification of the routine that decided to execute a Stop, and some indication of the location within a routine where the Stop occurred (some routine can have many types of error detection events). If a Stop occurs one can examine the routine for possibly more info, BUT, one should always check the GERROR file, as these Stops are frequently attended by an "intelligent error diagnosis" being written to the GERROR file by the offended-routine.

There are a class of numerical errors that occur when catastrophic numerical instability sets in upon the integration processes within GTOSS. These frequently show up as a Stop in certain Utility routines such as "VECNRM", and is usually a manifestation of the fact that numerics have gone-bad (ie floating point under/over-flows, zeroes in denominators where they shouldn't be, etc, etc). One very quickly learns to recognize the artifacts of this error, and if you examine the numeric results this is frequently preceded by pathological and erratic behavior and oscillations in numerical values, etc. **This is why it is mandatory to read the section in both the GTOSS and TOSS reference manuals pertaining to "Numerical stability".**

Making Your First Example GTOSS Run

To be concrete, we will make an example Space Elevator related run. Create a directory, say **/SE_Run/**. Make a copy of the executable programs **G**TOSS and **C**TOSS into this directory. Now, in the delivery data, there is a directory (folder) within which is an INGOSS and INCOSS files as shown below:

/Your_first_GTOSS_Run/ ← a Directory-folder
INGOSS (SE Lib) ← GTOSS Sample Run input file
INCOSS (SE Lib) ← CTOSS Post Processor input file

Move these two files to your directory **/ SE_Run /** and rename them to simply INGOSS and INCOSS. At this point your test run folder should look like this:

/ SE_Run /
INGOSS
INCOSS
GTOSSvH12
CTOSSvH12

Now, execute **G**TOSS by opening a Terminal/shell and navigating (current working directory) to **/ SE_Run /**, and entering the command **./**GTOSSvH12

This should start output data appearing in the Terminal shell; this output will consist of an initial set of run setup information, followed by a sequential synopsis of the time-history of the GTOSS solution as it unfolds. The run terminates at 100,000 sec, and you should see a "GTOSS Normal termination" enunciation. You can now examine the contents of / **SE_Run** / to see what has transpired. In your test run directory, you should see the OUTQUIK file (which you can open and examine with a text editor), and also the set of RDB files.

Next, execute the Post Processor **C**TOSS in a similar fashion to what you did to execute GTOSS (just use the same Terminal shell, since its "current working directory" is already set to / **SE_Run** /). After running CTOSS, you should *immediately* get an enunciation of Normal CTOSS termination. At this point a new set of files will have appeared in / **SE_Run** /; these will be files containing "column delimited data results" from the simulation.

Now, examine the CTOSS input file (INCOSS) by opening it in a text editor. Each (of the 7) output file invocation-blocks will start by specifying a CTOSS format. Look in the Quick Ref Use manual (under CTOSS formats), or the CTOSS User manual to see details about what the columns of data represent. Finally, with your favorite engineering graphing program, see if you can open and plot some variables that look to be interesting. If you do this, you will have successfully executed your first GTOSS simulation of the Space Elevator and derived actual useful practical results.

Last, examine the INGOSS file and try some variations in parameters; for example:

- Shorten or lengthen the run time; or,
- Output data less frequently, or more frequently; or
- Change the "libration perturbation". Note: since the run is configured to perturb the SE from a stable vertical configuration by a linearly distributed velocity perturbation profile along its length that culminates at a peak value of 2000 fps at the ballast (ie. SE upper body) mass, search the INGOSS file for the number 2000 to find how that perturbation was specified, then modify it to say 4000.

Making More Advanced Space Elevator Explorations

Included in the GTOSS delivery (/SE Dynamics Exploration Runs/) are a number of individual directories representing a total of 35 different SE runs. These runs can be conducted in exactly the same way as your initial example run (described above) was conducted, except now you will be executing complex simulations of all the various dynamically interesting aspects of the SE; the Space Elevator dynamic genres' addressed with these runs are:

SE Basic Tapered Ribbon Examination
SE Nominal Stress State Examination

SE Longitudinal Ribbon Modes
SE Transverse Ribbon Modes

SE Libration Dynamics

SE Transverse Wave Propagation
SE Stress Wave Propagation

SE Climber Liftoff
SE Climber Transit
SE Climber Transit Arrest
SE Climber Transit Resume

SE Aerodynamic Response

For each run genre, there is a Word file describing (multiple) related runs, special admonitions related to each run, and a diagram of each simulated configuration.

Basic graphical results for these runs can be found in the **SE Dynamics Handbook** (available on the SE Wiki).

For each **INGOSS** file, there is one (or more) corresponding INCOSS file(s), these being input streams for CTOSS post processing of the simulation runs; by using these INCOSS files, CTOSS will generate the output data that has been presented in the corresponding sections of the Dynamics Handbook. To execute these runs is quite simple, but to fluently build on and extend these run explorations requires a significant background in Space Elevator analysis as well as simulation experience with complex dynamical systems, *however, each run does represent a very convenient starting point to in fact proceed along these lines (thus saving the user significant time developing new explorations stemming from those already done).*

Admonitions

As the author of GTOSS who has served the engineering/aerospace tether community for over 30 years, providing formal class instruction in the use of GTOSS and tether dynamics, informal help to many users, GTOSS installation at enumerable facilities on enumerable different computer platforms and operating systems, general consulting and conducting design studies for many clients, I have made the following observations;

99% of the problems perceived by the user as being a bug in GTOSS, is in fact a user precipitated error; either an overt miscue in interpreting and/or entering physical system specification data and execution options, or, inattention to numerical convergence of integrated-time-domain dynamics solutions, or, a lack of understanding in interpreting results.

One of the most insidious (and common) problems that tend to plague new users is the failure to be ever vigilant in insuring the "Numerical Convergence" of the integrated solutions to the GTOSS equations. To this end, it is mandatory that a new user read and understand fully Section 4 of the GTOSS User Manual, and Section 7 of the TOSS User manual. Both non-converged as well as fully-diverged numerical solutions are easy to detect and remedy, but the user must become aware of the phenomenon in order to be vigilant in this regard.

Perusal of the "input skeleton files" (included in the delivery), and that contain the input data "item identification numbers" for every input item known to GTOSS, etc, can function admirably as a complete and concise GTOSS features expose' (actually to a fault, since EVERYTHING GTOSS does is represented there).

You may not like the GTOSS user interface (admittedly, it is a bit dated since GTOSS was conceived and written initially before bit-mapped monitors when all user-interaction was via text-line output, with no GUI's), but realize that the input specification problem of defining to the simulation code "an arbitrary number of bodies connected in arbitrary fashion, by an arbitrary number of tethers" is not a trivial user-interface or data specification problem. That admitted, GTOSS has been used by MANY scientific and engineering organizations in the hands of highly skilled and experienced engineers and scientists and has "passed muster", as well as passed "simulation reconciliation testing" against an assortment of other recognized tether dynamic codes. The point here is that one MUST read at least the basic documentation describing what GTOSS does, and how it does it; then they will be in a much better position to assess the nature of difficulties that are encountered, and look first to themselves for the answer rather than blame a code that is likely almost as old, or older than they are ☺ .

Recommended Reading

Included in the GTOSS delivery area are a number of .pdf files containing extensive documentation of GTOSS operation. In order of decreasing importance (with **Red** meaning MOST important).

GTOSS Reference Manual

At least a skim-read of Sections 1, 2, **4 (very important)**, 5, 6, 8, 10, 11, 12

TOSS Reference Manual

At least a skim-read of Sections 2, 3, 5, 6, **7 (very important)**, 8

Quick Reference Manual

This volume contains a potpourri of information, some of which is *unique* to this document, others of which has been duplicated (and shortened) from other documents for which frequent reference is made, but all of which has been found to be highly useful, whether setting up runs, or modifying code. This is the bread-and-butter day-to-day stalwart of the GTOSS documentation. *The first section in the Quick Ref is particularly valuable in interpreting results data as much of these results are referenced to the various "frame definitions" within GTOSS*. This also boils down concisely much of the information pertaining to the myriad of dynamics Initialization, Execution and Control options.

G/TOSS Equations Reference Manual

Provides detailed equation derivations, modeling assumptions, and the equivalence mappings between "analytical dynamics symbols" -to- "Fortran variable names". This is an abstruse dissertation, the full understanding of which would require significant experience in advanced dynamics; that said, it does give one an appreciation and overview of resides within GTOSS. The source code files contained in the directory /**A_HDR**/ are also vital to understanding the definition of results data made available to the user via DTOSS and CTOSS; in particular, the header files,

EQU_HOST.i	Host simulation (TOSS Object 1, the Ref Point) data items
COM_RPS.i	the State of the TOSS Ref Point and items related to a wide scope
COM_ALL.i	Universal parameters used by all
EQU_OBJI.i	Non-Body-specific data items related to ALL TOSS Bodies
EQU_TOSS.i	data items related to ALL TOSS

In the files below, disregard the lower case "r" and "I" preceding variable names: the uppercase characters are the ones being defined and used in actual code.

Finite_Solution_Data_Structure.i	Finite Tether *Object* data items
TOSS_Object_Data_Structure.i	TOSS Body *Object* data items

Interface Control Document

Many concepts which the other documentation has assumed to be common knowledge (to the user), are in fact introduced and defined only in this manual.

DTOSS Reference Manual
CTOSS Reference Manual
UTOSS Reference Manual
RTOSS Reference Manual
Skim-read these for indoctrination only.

VTOSS Reference Manual

Useful read if you intend to validate your installation or do user-modifications to GTOSS.