GATS Software Documentation GATS-1994-03

Scientific Software System (S^3)

Version: 2.0
User's Guide

John Burton
Larry Gordley

September 29, 2005

GATS, Inc.
11864 Canon Blvd. Suite 101
Newport News, VA. 23606
(757) 873-5920

Contents

List of Figures

1 Introduction

There are many facets to scientific modeling, analysis and data processing. Most require complicated mathematical procedures that will be be refined through research efforts spanning several years. These efforts often involve scientists and software developers from a variety of research centers. The purpose of the Scientific Software System (S^3) is to provide a flexible basis for research software development that can be rapidly adapted to the needs of both individual and group efforts, which includes orderly evolution of complex software.

Research scientists must have detailed control of their software. Science is by nature experimental, which translates to fast and frequent changes. These two facts are often neglected by programming purists who insist on rigid developmental rules. Somebody once stated that "Creative minds are seldom tidy". But large untidy programs, even if spawned by creativity, will eventually become a severe burden to maintaining that creative momentum. So scientists are torn between the excitement of racing to an answer and the future's demand for maintaining order, structure, and documentation. S^3 solves this dilemma by supplying the ability to do both.

The S^3 package is a software development tool that encourages the use of top-down software design techniques and modular programming concepts. S^3 combines the flexibility and rapid prototyping of an interpreted programming language (such as BASIC) with the power and speed of a compiled programming language (such as FORTRAN or C). Central to S^3 is the top-down design concept that each task can be divided into a series of sub-tasks, and each sub-task can be further divided into a series of sub-tasks. For software development this implies that each of the higher level modules (tasks) should be essentially a series calls to lower level modules (sub-tasks). In essence an S^3 program is simply a list of calls to executable modules. The order in which the modules are called can be dynamically changed at run-time.

2 Basic Design Overview

The primary S^3 objectives are:

1. Easy to use. After setup, the user needs to know the syntax for defining a variable and calling a module, and little else.
2. Easy and reliable modification of the system, allowing efficient group development efforts.
3. Modification without high level code change, aiding in configuration control.

4. Easy maintenance of modularity throughout code development, facilitating a continuing orderly evolution.
5. Simple use of complex modules.
6. Easy incorporation of new analysis techniques.
7. Simple interface to graphical analysis tools.
8. Strong inducement to use structured programming methodologies.
9. Simplification of understanding and documentation.

These objectives can be accomplished with strict definition and control of data flow and module interface, and flexible definition of processing logic and data structures. The method used to achieve these goals is based on four major components; the Executable Module Table (EMT), the Parameter Definition Table (PDT), the Process Configuration Tables (PCTs), and the Process Control Program (PCP). The EMT defines the executable modules that will be used by the system, including type of routine and the name of the object file.The PDT defines the constants and datasets that will be used by the system. Similarly, the PCTs define the order in which modules are executed and which datasets are passed to a given module. The PCP is the program that ties everything together. It dynamically loads the modules specified in the EMT, allocates memory for the datasets specfied in the PDT, and controls the module execution sequence based on the information provided by the PCT's. The use of this architecture provides additional desirable research features:

1. Module execution can be easily reconfigured, either before or during execution by simply modifying the PCT's.
2. Program control can be exercised from any module in the system.
3. Interactive graphical control can be realized simply by inserting independently developed and tested graphics modules.
4. The user can install his own routines by simply specifing the executable module in the EMT file and modifying the appropriate PCT file(s) before execution.
5. Modules can be written in various languages (currently limited to C and FORTRAN).

2.1 EMT - Executable Module Table

The EMT file contains the names of all the functions and subroutines (and associated filenames) that are called directly from the PCT level of an S^3 program. Also included in the EMT are the names of any library and object files that support those functions and subroutines. All the exectuable modules listed in the EMT file are collected into a shared library whose members are dynamically loaded as needed during program execution. An example of an EMT file is included below in figure 1.

```
!
! FORTRAN Subroutines used
!
proc:fs: fillarray;
proc:fs: fprintarray;
!
! C Subroutines used
!
proc:cs: cprintarray;
proc:cs: dump;
!
! Sub-PCTs used
!
spct: test2: test2.pct;
!
! Support Libraries used
!
objs: fillarray.o, fprintarray.o, ./csubs/cprintarray.o, ./csubs/dump.o;
libs: libsupport.a;
```

Figure 1: Example of a simple EMT file

2.2 PDT - Parameter Definition Table

The PDT file defines the type of data, number of dimensions, and size of each dimension for all datasets that are used by an S^3 program. A dataset **must** be defined in the PDT file before it can be used by the program. The PDT can be thought of as a large (FORTRAN) common block of global data structures, but with critical distinctions:

1. The data structures are held in a file, eliminating the necessity of code change to a common block.

2. The data structures can be accessed only through the calling parameters associated with routines specified in the PCT's.

This forces preservation of modularity while permitting rearrangement of logic and data structures with little or no code change.

The PDT file syntax allows the definition of constants which can be later used to dimension datasets. Each dataset can have zero or more dimensions, with zero dimensions indicating a simple dataset containing a single value. Additionally each dataset can be initialized within the PDT file or from an S^3 control file. An example PDT file, shown in figure 2, demonstrates the basic syntax of a PDT file. Further details concerning the syntax of a PDT file are provided in the S^3 Reference Manual.

```
!
! Constants used to dimension the datasets
!
CONST:  m1     =       5;
CONST:  m2     =       4;
CONST:  m3     =       3;
!
! Definition and initialization of a single dimensioned array
!
DREAL:  arr1     : m1 = {0.0,1.0,2.0,3.0,4.0};
!
! Definition and initialization of a two dimensional array
!
REAL:   arr2     : m1,m2 = {0.0,0.1,0.2,0.3,0.4,
                            1.0,1.1,1.2,1.3,1.4,
                            2.0,2.1,2.2,2.3,2.4,
                            3.0,3.1,3.2,3.3,3.4};
!
! Definition and initialization of a three dimensional array
!
REAL:   arr3     : m1,m2,m3 = {0.00,0.01,0.02,0.03,0.04,0.10,0.11,0.12,0.13,0.14,
                               0.20,0.21,0.22,0.23,0.24,0.30,0.31,0.32,0.33,0.34,
                               1.00,1.01,1.02,1.03,1.04,1.10,1.11,1.12,1.13,1.14,
                               1.20,1.21,1.22,1.23,1.24,1.30,1.31,1.32,1.33,1.34,
                               2.00,2.01,2.02,2.03,2.04,2.10,2.11,2.12,2.13,2.13,
                               2.20,2.21,2.22,2.23,2.24,2.30,2.31,2.32,2.33,2.34};
!
GROUP:agroup:arr1,arr2,arr3,m1;
```

Figure 2: Example of a simple PDT file

One construct that should be noted in the example PDT file, figure 2, is the *group* construct. *group* associates a name with an ordered group of datasets and / or constants, which can later be used to pass those datasets / constants to an executable module.

2.3 PCT - Process Configuration Tables

In contrast to the PDT, which remains unchanged after initialization, the PCT's are actively involved in the control of the program's execution. Each PCT consists of an *Execution Order List* (EOL), an *Execution Order Index* (EOI) and an *Execution Entry List* (EEL). Each Execution Entry either specifies a call to an executable module or transfer of control to another PCT. This structure is analogous to the multi-level menu interfaces used by many popular software packages. In these packages, each menu option either performs some operation or produces a sub–menu. The order in which the entries of a PCT are executed is controlled by the EOL and EOI. To provide flexibility, the EOL and EOI for any PCT can be modified at any time during program execution. Like the PDT, the PCT's are initialized from the appropriate PCT files during program startup.

```
EOL=b,a,c,a,d,z;                        ! Execution Order List (EOL)
EOI=0;                                  ! Execution Order Index (EOI)

a:test2;
b:fprintarray   :agroup,m2,m3;
c:cprintarray   :arr1,arr2,arr3,m1,m2,m3;
d:dump         :arr1,arr2,arr3,m1,m2,m3;
```

Figure 3: Example of a simple PCT file

```
EOL=a,b,c,z;                    ! Execution Order List (EOL)
EOI=0;                                ! Execution Order Index (EOI)

a:fillarray    :arr1,arr2,arr3,m1,m2,m3;
b:fprintarray   :agroup,m2,m3;
c:cprintarray  :arr1,arr2,arr3,m1,m2,m3;
!d:dump         :arr1,arr2,arr3,m1,m2,m3;
```

Figure 4: Example of a simple sub-PCT file

The example PCT file given in figure 3 shows the the basic syntax of a simple PCT file. The PCT file contains a verbal description of the purpose of the PCT, the EOL, the EOI, and a list of entries. The EEL is constructed from the list of entries on program startup. The characters comprising the EOL correspond to the first letter in each line of the list of entries. This first character is referred to as the entry label. The letter "z" is used to indicate that execution in this PCT should be halted at this point and control should be passed to the calling or parent PCT. If there is no parent PCT then execution of the program is halted. The EOI is an index into the EOL, with zero (0) being the index of the first character in the EOL. The example in figure 3 will execute the specified routines in the following order:

```
FPRINTARRAY  :arr1,arr2,arr3,m1,m2,m3;
FILLARRAY    :arr1,arr2,arr3,m1,m2,m3;
FPRINTARRAY  :arr1,arr2,arr3,m1,m2,m3;
CPRINTARRAY  :arr1,arr2,arr3,m1,m2,m3;
CPRINTARRAY  :arr1,arr2,arr3,m1,m2,m3;
FILLARRAY    :arr1,arr2,arr3,m1,m2,m3;
CPRINTARRAY  :arr1,arr2,arr3,m1,m2,m3;
DUMP         :arr1,arr2,arr3,m1,m2,m3;
```

The next set of characters after the entry label specify what type of routine each entry is. In the example, `FPRINTARRAY` and `FILLARRAY` are FORTRAN Subroutines, and `CPRINTARRAY` and `DUMP` are C Subroutines. Other possibilities are described in the S^3 Reference Manual.

It should be noted that when a *group* name is encountered in a PCT, the PCP (described below) simply substitutes the datasets / constants associated with that group (in the order given in the PDT file) and then passes them to the executable module.

2.4 PCP - Process Control Program

The Process Control Program is the system manager. The PCP consists of two primary routines, one to initialize the environment which:

1. Parses the command line options to determine the operating mode of the system.
2. Reads the EMT file, ensures that the executable module is still current relative to its source code, and creates dynamic links to those modules.
3. Reads the PDT file, creates and initializes the required datasets.

4. Reads the PCT files and creates an execution tree whose interior nodes are PCTs and whose exterior or terminal nodes are executable modules.

The second primary routine traverses the PCT tree created during initialization in the order specified by the EOL and EOI, and executes the appropriate modules.

For example, assuming we have an S^3 application program named `example` and the EMT, PDT and PCT files shown in figures 1, 2, and 3 are named `example.emt`, `example.pdt` and `example.pct` respectively, a typical invocation of the program

```
s3 -b example
```

would result in the following actions:

1. Set the operating mode for `example` to check for dataset bounds violations (see the S^3 Reference Manual for more about operating modes).
2. Read EMT file `example.emt` :
   (a) Compare the timestamps for:
      - `fillarray.o` vs. `fillarray.f`
      - `fprintarray.o` vs. `fprintarray.f`
      - `./csubs/cprintarray.o` vs. `./csubs/cprintarray.c`
      - `./csubs/dump.o` vs. `./csubs/dump.c`

   (b) Recompile any modules that are out of date.
   (c) Dynamically load the executable modules so that they may be called by the PCP.
3. Read PDT file `example.pdt` :
   (a) Create *constant* entries for `m1, m2,` and `m3`.
   (b) Create *dataset* entries for `arr1, arr1,` and `arr3`.
   (c) Allocate memory for datasets `arr1, arr2,` and `arr3`.
   (d) Initialize `arr1, arr2,` and `arr3` to the specified value.
4. Read PCT file `example.pct`: and create a PCT tree containing 1 PCT entry and 4 executable entries.
5. Start executing `example.pct`. Repeat the following until EOL(EOI) equals "z".

(a) Get the character EOL(EOI).

(b) Call routine with label given by EOL(EOI).

(c) Check datasets for bounds violations.

(d) Increment EOI.

This is shown diagrammatically in figure 5.

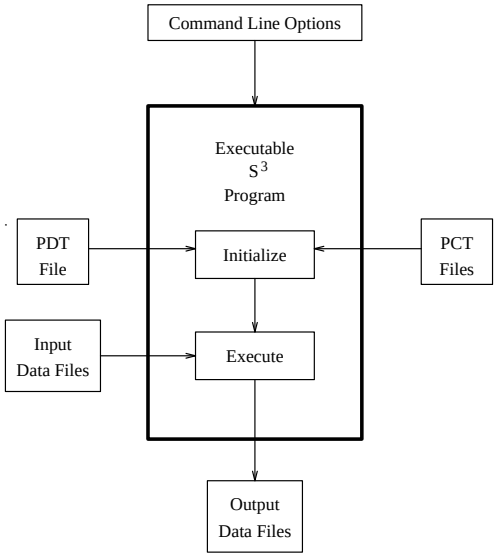

Figure 5: Executing an S^3 Program

The creation of the executable program `example`, as shown in figure 6, is accomplished by simply typing `make` which in turn performs a three step process. First the `setup` utility program reads the PDT and PCT files to obtain certain information about the routines `fillarray, fprintarray, cprintarray,` and `dump`. Next, this information is inserted into the main program (PCP) file which is then compiled along with the files `fillarray.f, fprintarray.f, cprintarray.c,` and `dump.c`. Finally, these five files are linked together along with the S^3 library to create an executable program. These steps are all controlled by the file `Makefile` which controls the `make` process.

Once the executable program has been created, the user has a limited freedom to modify how the program executes without having to recreate (re-`make`) the program. The user may:

- Change the EOL or EOI line of the PCT file, modifying the order in which the routines are executed, or which routines are executed.
- Remove a routine from the PCT file.

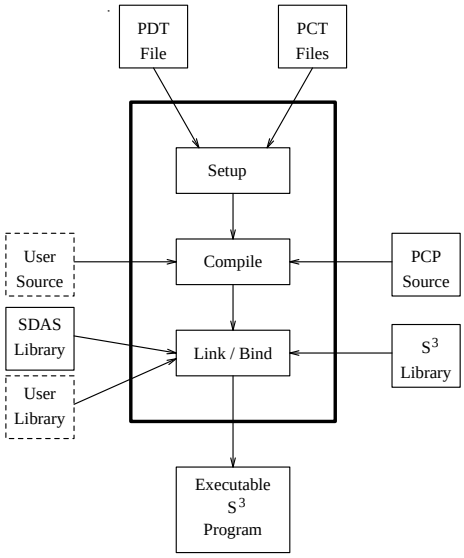

Figure 6: Creation of an S^3 Program

- Change the parameters being passed to a routine.

Any other modifications will generally require a re-`make`.