Unit testing is an established practice in professional software development. However, in high-performance computing (HPC) with its scientific applications, it is not widely applied. Besides general problems regarding testing of scientific software, for many HPC applications the effort of creating unit tests on legacy code can be make use of the existing infrastructure and extract test data from the running application. FTG generates code for serializing and storing a subroutine’s input data and inserts this code temporarily into the subroutine (capture code). In addition, FTG generates a basic test driver which loads this data and runs the subroutine (replay code). Meaningful checks and test data modification needs to be added by the developer.

**Source Code Analysis and Generation**

In a first step, the user has to provide assembler files from which FTG builds a call graph starting from the subroutine under test (SUT) (Step 1) from which FTG builds up a call graph starting from the subroutine under test (SUT) (Step 2). The code for capturing all the needed variables is automatically generated using static code analysis (Step 3) and inserted by FTG into the original application code (Step 4). The code analysis is conducted to find out which variables are actually needed by the subroutine. These are the input variables but also global or module local variables are considered. Finally, a basic test driver for loading the input data and running the subroutine in isolation is generated (Step 5). All of the generated code can be customized by templates.

**Runtime**

Now, the user can compile and run the original application, now containing the capture code, to extract a consistent set of input data for the SUT. The default time of capturing (e.g. the 1st or any 9th execution of the SUT) is defined by the template, but can easily be changed in the generated code. Afterwards the test driver can execute this captured data. When the first replay was successful, the user can go on and extend the test driver with meaningful checks or add additional tests by calling the SUT again with altered data.

**The Tool**

**General Function of FortranTestGenerator**

- **Application with capturing code**
- **Application with replaying code**
- **Test Driver with capture code**
- **Test Driver with replay code**
- **Capture & Replay with MPI**

**Example**

- **Capture** The code generated by FTG (red boxes) inserted into the existing subroutine.
- **Replay** Test driver generated by FTG.

**Source Code Analysis**

All of the code generated by FTG is based on customizable templates. So, one can for example...

- replace the serialization framework
- include and initialize additional libraries
- make use of a unit testing framework
- add some standardized checks
- choose between embedding into the main application/a test suite, or generating stand-alone test programs

**Code Templates**

- **Configuration file**
  - Location of source and assembler files
  - Exclude modules, routines or variables
- **Code templates**
  - (optional, default templates available)
- **Commandline interface**
  - For example to generate capture code

**User Interface**

- **FRONTEND**
- **APPLICATION**
- **SOURCE CODE**
- **EXECUTION**
- **ACCELERATOR**
- **RUNTIME**

**FACTS**

- **Written in Python**
- **Templates based on the Cheetah Template Engine**
- **Works with Fortran® or later**
- **Needs GCC assembler files for analysis**
- **http://github.com/fortesg**

**REFERENCES**