The GRASS Test Suite

Design and Usage

Overview

1. Motivation
2. Goals
3. Functionality
4. Validation and dependencies
5. Handling the test suite
6. Perspectives

Motivation to create the test suite

- Tests are essential for software development
- There is no functional test suite for GRASS 6 right now
- Provide easy error detection
- Error reduction
- Encourage the user acceptance with higher stability of GRASS
- Fastening the development process of GRASS

Goals of the test suite

- Platform independent test framework
- Test most of the GRASS modules of correct runtime behaviour
- Validation of the generated output
- Providing a standardised environment for easily implementation of tests
- Creation of clear and intelligent summaries and log files
- Supporting the development process

Parts of the test suite

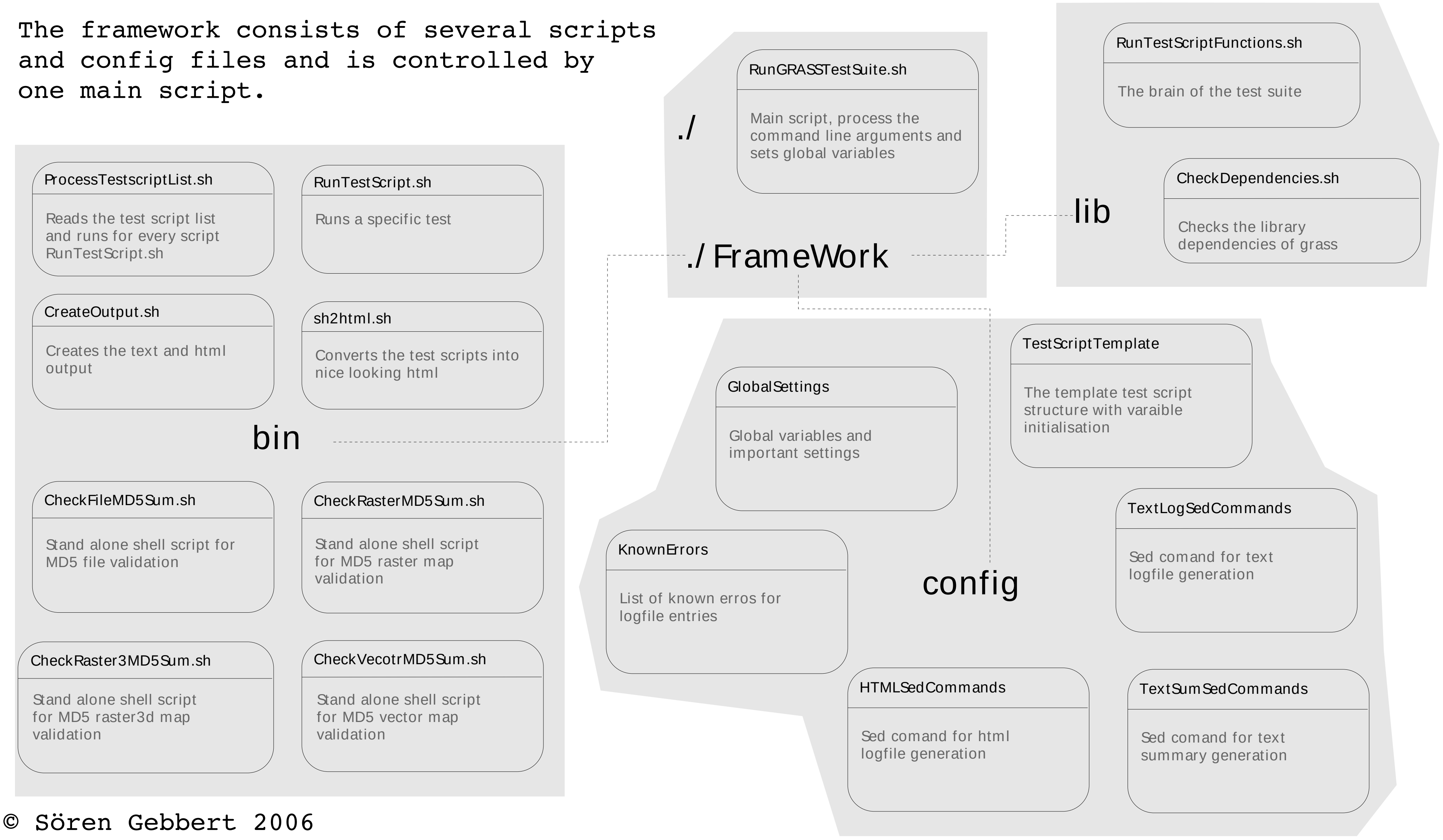
- Framework
  - Provides a standardised environment for tests and manages the execution and evaluation of tests
- Test scripts
  - Describe the tests and there behaviour
  - Have an specific syntax
- Test location
  - A small grass test location with raster, raster3d, vector and database data

The framework

- Provides are standardised environment for GRASS
- Consists of a collection of shell scripts and Unix programs
- Must be executed within GRASS
- Can handle one, a specific set or all available tests
- Takes care of validation and erasing of module output (raster/raster3d maps, vector maps and files)
- Can handle unit and integration tests
- Supports memory checks of every module via valgrind
- Checks grass library dependencies before module execution
- Crates html and text output
- All scripts and config files are located in the directories FrameWork/bin, FrameWork/lib and FrameWork/config

Structure of the framework

The framework consists of several scripts and config files and is controlled by one main script.


./
RunGRASSTestSuite.sh
Main script, process the command line arguments and sets global variables
./ FrameWork
lib
RunTestScriptFunctions.sh
The brain of the test suite
CheckDependencies.sh
Checks the library dependencies of grass
bin
ProcessTestscriptList.sh
Reads the test script list and runs for every script RunTestScript.sh
RunTestScript.sh
Runs a specific test
CreateOutput.sh
Creates the text and html output
sh2html.sh
Converts the test scripts into nice looking html
CheckFileMD5Sum.sh
Stand alone shell script for MD5 file validation
CheckRasterMD5Sum.sh
Stand alone shell script for MD5 raster map validation
CheckRaster3MD5Sum.sh
Stand alone shell script for MD5 raster3d map validation
CheckVecotrMD5Sum.sh
Stand alone shell script for MD5 vector map validation
config
GlobalSettings
Global variables and important settings
TestScriptTemplate
The template test script structure with varaible initialisation
KnownErrors
List of known erros for logfile entries
TextLogSedCommands
Sed comand for text logfile generation
HTMLSedCommands
Sed comand for html logfile generation
TextSumSedCommands
Sed comand for text summary generation

The test script concept

- Test scripts describe properties and execution of tests
- Test scripts have bash syntax and fixed variables
- Within test scripts, the modules, command line arguments, MD5 checksums and library dependencies are declared
- Test scripts are handled as shell scripts, the hole bash functionality can be used within them
- They can handle different modules with multiple output (r3.to.rast, r.terraflow)
- Test scripts are located in the “Tests” directory and distributed to the directories: database, display, general, imagery, raster, raster3d, scripts and vector
- Each test directory contains a list file with entries for every test in this dir
- This list provides an easy and simple way to define module dependencies

The test script concept

Simple Example

```
##########################################

#Title
Title="g.version Test"

#A description of the test
Description="Basic function test of g.version "

#The number of tests
NumberOfTests=3

#The module which should be tested
Module[0]="g.version"
Module[1]="g.version"
Module[2]="g.version"

#the module options
ModuleOptions[0]=""
ModuleOptions[1]="-c"
ModuleOptions[2]="-b"
```

The title of the test, is written to the html and text output

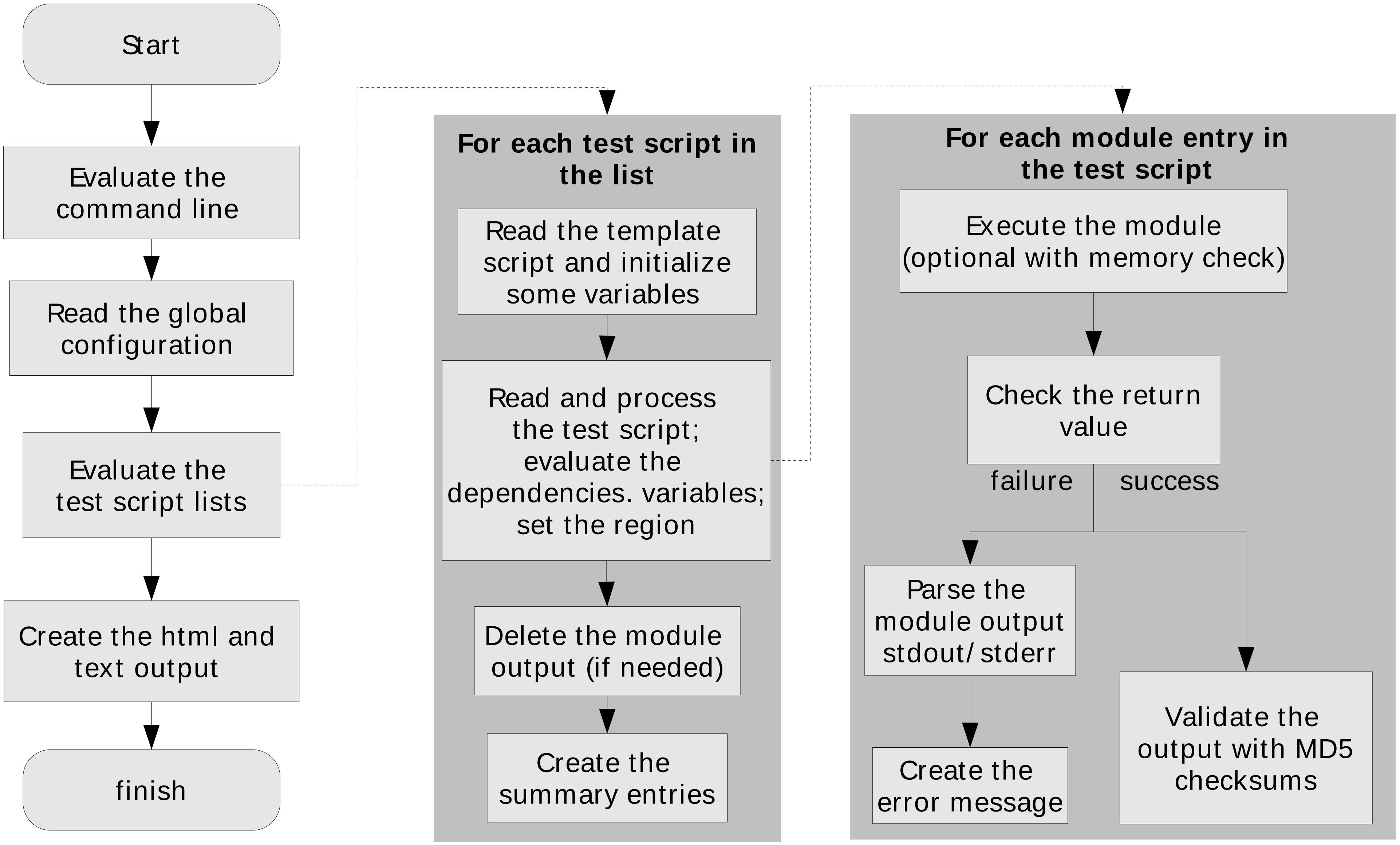
Description of the test, is written to the html and text output

The number of tests in this script to process

List of modules which should be executed

The command line arguments for every module

Functionality of the test suite


Start
Evaluate the command line
Read the global configuration
Evaluate the test script lists
Create the html and text output
finish
For each test script in the list
Read the template script and initialize some variables
Read and process the test script; evaluate the dependencies. variables; set the region
Delete the module output (if needed)
Create the summary entries
For each module entry in the test script
Execute the module (optional with memory check)
Check the return value
failure
success
Parse the module output stdout/ stderr
Create the error message
Validate the output with MD5 checksums

The validation system

- Validation of module outputs with
  - Reference data within the test location or
  - MD5 checksum of the reference data
- Validation is supported for
  - Raster maps, raster3d maps, vector maps and normal files (output of r.out.vtk ...)
- The data validation is based on (truncated if floating point) output of:
  - r.out.ascii, r3.out.ascii und v.out.ascii

GRASS modules within the test suite

- The following modules are used within the test suite
  - r.out.ascii, r3.out.ascii, v.out.ascii
  - g.remove, g.region, g.parser, g.tempfile, g.version
- This creates a dependency of the functionality from these modules!

Handling of the test suite by examples

1. Download, installation and start of the GRASS test suite
   a) Download the latest test suite from
      www-pool.math.tu-berlin.de/~soeren/grass/GRASS_TestSuite
   b) Unpack the tar.bz2 file in any directory
   c) Enter the GRASS_Testsuite directory and execute StarGRASS.sh
   d) That's all!

Handling of the test suite by examples

2. Execution of one and several tests
   a) Execution of g.version test
   b) Execution of database (db.*) tests
   c) Execution of general (g.*) tests
   d) Execution of Raster (r.*) tests
   e) Execution of all available tests
   f) Execution of g.version test with memory check

Handling of the test suite by examples

3. Handling of the html output
   a) The html summary
   b) The test suite logfile
   c) The source code of the processed test
   d) The output of the memory checker

Handling of the test suite by examples

4. Implementing a simple test
   a) Use an existent test as skeleton
   b) Look for extra options in the TestScriptTemplates file
   c) Execute the test
   d) Create an entry in the list file

Handling of the test suite by examples

5. Implementation of a complex test of r.slope.aspect
   a) Use an existing test as skeleton
   b) Look for extra option in the TestScriptTemplates file
   c) Handling of multiple Output
   d) Creation of the MD5 checksums
   e) Execute the test
   f) Create an entry in the raster.list file

Perspectives

- Provide a self test mechanism for all modules used within the test suite
- Integration of the test suite in the GRASS build system
- Encourage the acceptance of the test suite

That's all!