Simonas Joris Samaitis
University of Vilnius
26/09/2011

Regression Testing Suite for Condition Data Storage

Purpose :

Automated testing of condition data formats for backward and forward compatibility. Writing and reading data between different versions and architectures of CMSSW framework. More detailed information about the suite functionality can be found in regression testing suite presentation.

Contents :

Payload classes :

Class dependency tree

```
TestPayloadClass
        Primitives
        DataStructs
        Inheritances
Primitives :
        int testInt;
        long int testLongInt;
        double testDouble;
        std::string testString;
        enum TestEnum { A =3, B, C= 101, D, E, F};
        TestEnum testEnum;
        typedef int TestTypedef;
        TestTypedef testTypedef;
DataStructs:
        TestStruct testStruct;
        struct TestStruct
                std::string testStructString;
                int testStructInt;
        Color tmpColor;
        struct Color
                int r;
                int g;
                int b;
        std::map<std::string, std::vector<Color> > testTripletMap;
        std::list<std::string> testStringList;
        std::pair<std::string, int> testPair;
        std::set<char> testSet;
        std::vector<std::string> testStringVector;
        std::vector<int> testIntVector;
Inheritances:
class TestData
        int commonInt;
        std::vector<std::vector<int> > commonIntVector2d;
class TestInheritance : public TestData
        std::vector<std::string> dataStringVector;

TestPayloadClass :
        Primitives primitives;
        DataStructs dataStructs;
        Inheritances inheritances;
bool DataToFile(std::string fname); //Writes the payload data to file (debug)
```

Every class has == and != operators implemented for testing all included data types and structures.
Includes.h has all the headers and defines needed for payload classes.

Classes for accessing database :

TestFunct :

```
bool Write(std::string mappingName, int payloadID);
bool Read(std::string mappingName);
bool ReadAll();
bool CreateMetaTable();
bool DropTables(std::string connStr);
bool DropItem(std::string mappingName);
```

Those functions are used to write the payload to DB. They can be accessed only using testCompat with parameters :
usage : testCompat [arguments]
-c creates new TEST_SEED and metadata tables

-d [mappingName] drops item

-D drops all items

-r [mappingName] reads item

-R reads all items

-w [mappingName] -s [seed] writes item

afterwards the following arguments must be supplied :

-A [authentication path] –C [connection string]

Example s:

```
testCompat –c –A auth/path/here –C  oracle://database@username
testCompat –r itemname   –A auth/path/here –C  oracle://database/username
testCompat –w itemname –s 10  –A auth/path/here –C  oracle://database/username
```

It is possible to display the list of usage cases and possible arguments by executing “ testCompat –h”

Python script that executes the classes : testRegression.py

This scripts is used to manipulate version_table , results_table and test status databases(described in later chapter) and to run test executables in specific order to test the regression. Every test sequence compares two releases – candidate release (provided as an argument) and reference release(provided as a second set of arguments) or a set of releases (defined in the version_table). Following is the test sequence used :

```
Set environment for reference release
Drop all tables with reference release
Create MetaData tables with reference release
Write data with reference release
Set environment for candidate release
Read data with candidate release *
Write data with candidate release *
Read data with candidate release *
Set environment for reference release
Read data with reference release *
```

Segments marked with * represent key parts of the test, they are logged in test_status table.

Usage and command line arguments :

-c (-s) creates descriptor(status) db schema"

-d (-s) drops descriptor(status) db schema. Optional : -R [release] -A [arch] to drop single entry"

-w writes data to db. Goes only with -R [release] -A [arch] -P [path]"

-r (-s) reads contents of descriptor(status) db"

-t (-o) runs test. Goes only with -R [release] -A [arch] -P [path] -S [seed] -L [label]"

-(o) specifies reference release. supply additional parameters --R [refRelease] --A [refArch] --P

[refPath]

-L [label] marks the name of the test sequence used. Test sequences are described in the following chapter.

Usage examples :

```
python testRegression.py -c
python testRegression.py -d -s
python testRegression.py -t -R CMSSW_4_2_8 -A slc5_amd64_gcc451 -P home/myDirectory
python testRegression.py -t -o -R CMSSW_4_2_8 -A slc5_amd64_gcc451 -P home/myDirectory
--R CMSSW_3_8_7 --A slc5_ia32_gcc434 --P  /afs/user/test
python testRegresssion.py -h displays possible arguments
```

Test sequences descriptor file

Test sequences are defined in sequences.xml file, which has to be bundled with testRegression.py execution script. Structure of the file :

```
<xml>
   <test name = "[label]">
```

```
        <init>
           <command exec =”[command]” env=”[environment]” </command>
           <command -/other commands/- ><command>
        </init>
        <sequence>
           <command exec =”[command]” env=”[environment]” Rname = “[status label]  </command>
           <command - / other commands / -> </command>
        </sequence>
        <final>
           <command exec =”[command]” env=”[environment]” </command>
           <command -/other commands/- ><command>
        </final>
    </test>
</xml>
```

Each test must have <init>, <sequence> and <final> branches. <init> describes commands used to set up the initial values used in the test, status of the commands executed is not logged in the status_table, and final defines finalizing commands of the test.
<sequence> defines the core of the test, and every command used must return a value indicating the success or failure of the test. Return value of 0 marks successful test, other values indicate failure.

[label] of the test must be the same as used in execution of the testRegresssion.py Each test must have a unique label.
[command] defines executable with path and parameters to be executed
[environment] is the environment in which the command must be run, either of reference release - “ref” or “cand” – of candidate release.
[Rname] stands for the label of the return code from command executed. It is written in status_table and displayed in webApp.

Additional tables used by webApp.py and testRegression.py :

Version_table

Version_table holds a list of reference releases that are tested in pair with candidate release.

ID	RELEASE	ARCH	PATH
NUMBER	VARCHAR2(50)	VARCHAR2(30)	VARCHAR(255)
Number of entry	Name of reference release	Architecture of reference release	Path to reference release

Table also has a primary key PK_ID of pair (RELEASE, ARCH)

Test_status

Test_status holds data associated with test runs and log data for each test sequence

ID	RUNID	RDATE	LABEL	T_RELEASE	T_ARCH	R_RELEASE	R_ARCH	LOG
NUM	NUM	DATE	VCH2(20)	VCH2(50)	VCH2(30)	VCH2(50)	VCH2(30)	CLOB
No. of entry	Num. of run	Time stamp	Name of test	Candidate release	Candidate architecture	Reference release	Reference architecture	Logfile data

Table also has a primary key PK_ID2 of triplet (ID, RUNID, LABEL)

Test_results

Test_results table holds status information for each test pair(candidate release, reference release). This information is displayed in webApp.

RID	ID	LABEL	NAME	STATUS
NUMBER	NUMBER	VARCHAR(20)	VARCHAR(100)	NUMBER
Number of entry	ID of entry in test_status	Name of test	Name of status to display in webApp, [Rname] from xml	Execution result (return code)

Sequences

Sequence table defines sequences for automatic incrementing of ids in test_status table.

LABEL	ID	RUNID
VARCHAR(20)	NUMBER	NUMBER
Name of test	Number of entry	Number of Run

Structure of test log files :

%%% - start/end of test

- start/end of test sequence
** - start/end of command

Note for webApp.py : to use webApp, please create folders named „sessions“ and „logs“ in the same directory as webApp.py