Regression testing in the TOTEM DCS

F Lucas Rodríguez[1], I Atanassov[1], P Burkimsher[1], O Frost[1], J Taskinen[2] and V Tulimaki[2]
on behalf of the TOTEM collaboration
[1] CERN, CH-1211 Genève 23, Switzerland
[2] Savonia Ammattikorkeakoulu, Finland

Abstract.
The Detector Control System of the TOTEM experiment at the LHC is built with the industrial product WinCC OA (PVSS). The TOTEM system is generated automatically through scripts using as input the detector Product Breakdown Structure (PBS) structure and its pinout connectivity, archiving and alarm meta-information, and some other heuristics based on the naming conventions. When those initial parameters and automation code are modified to include new features, the resulting PVSS system can also introduce side-effects.

On a daily basis, a custom developed regression testing tool takes the most recent code from a Subversion (SVN) repository and builds a new control system from scratch. This system is exported in plain text format using the PVSS export tool, and compared with a system previously validated by a human. A report is sent to the developers with any differences highlighted, in readiness for validation and acceptance as a new stable version.

This regression approach is not dependent on any development framework or methodology. This process has been satisfactory during several months, proving to be a very valuable tool before deploying new versions in the production systems.

1. Introduction
The TOTEM (total and elastic measurements) experiment at CERN [2, 3, 4, 6] measures the total cross section of the proton and also monitors accurately the LHC luminosity [1]. To do this, TOTEM is able to detect particles very close to the LHC beams, after their collision in the interaction point.

TOTEM consists of three subdetectors. They are the 'Roman Pot Stations' (RP), the 'Cathode Strip Chambers' (CSC) Telescope 1 (T1) and the 'Gas Electron Multipliers' (GEM) Telescope 2 (T2). The T1 and T2 detectors are located on each side of the CMS interaction point in the very forward region, but still within the CMS cavern. Two Roman Pot stations are located on either side of the interaction point, at 220 meters and 147 meters, inside the LHC tunnel. Each Roman Pot station consists of two groups of three Roman Pots separated by a few meters, as shown in Figure 1.

PVSS [8] is a control system development tool structured around the concept of datapoints (DPs). Datapoints can considered as variables similar to a `C` language `struct` stored in a persistent database. Most of the PVSS scripting capabilities focuses on reading, writing and configuring datapoints.

All the LHC experiments' controls systems use PVSS and the JCOP framework [9] maintained by the EN-ICE group at CERN.

The TOTEM Detector Control System (DCS) team developed a set of automation scripts and a `C#` tool to generate those datapoints in a reliable and unified way [5, 10, 11]. This makes it possible to reconfigure

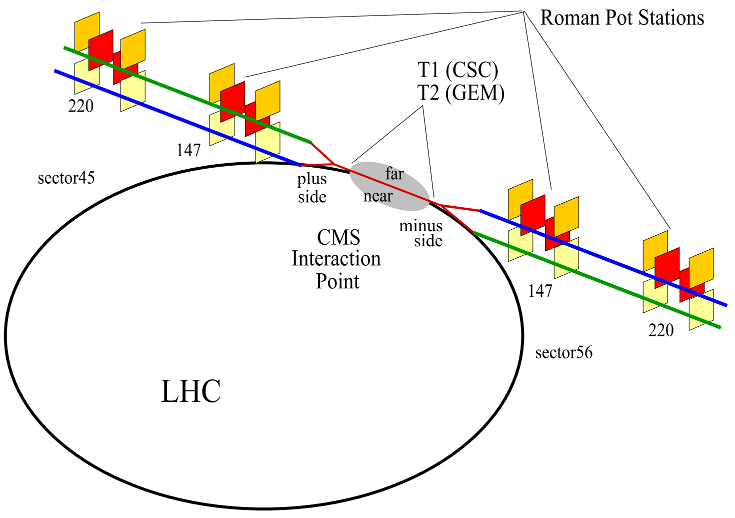


Figure 1. TOTEM locations

of the whole system within hours (within one working day) and a homogenizes the behaviour among subsystems.

2. Motivation

This paper presents the outcome of the effort spent to improve the reproducibility and understanding the final system changes during the project evolution. The purpose of our novel regression tool is to provide an easy way to evaluate the consistency of the final project with an earlier reference project which is considered to be working correctly.

Our custom code for the automated generation of datapoints is very reliable, however there is always need for new features and minor changes to deal with ongoing project maintenance. Those modifications somehow have undesirable side-effects in other TOTEM detectors or parts of the code. The TOTEM DCS group was faced with a software Quality Assurance issue.

Existing systems and tools have several limitations and cannot be used directly our our regression tests. The PVSS tool `pvss00ascii.exe` can export all datapoints configuration into a plain text proprietary format. The resulting file has the extension `.dpl`. Most of the LHC teams for controls use this file as an exchange of information between the production and developments systems.

However many times these files include too much information. There are many timestamps that would be better if they were not transferred to the final system. Also many datapoints linked to OPC (OLE for Process Control) servers [12] are marked as having an invalid status (as in the development environment it is not possible to replicate all the hardware components), and there are many other parameters that should not be transferred among systems. The outcome is clear, using PVSS `.dpl` as an exchange mechanism is not optimal and can lead to many problems. Those `.dpl` files cannot even be directly compared between versions, as all the timestamps change. Commercial `diff` tools notice all the changes and a cleaning step would be needed before comparing files.

The previously developed TOTEM automation tools are detailed in Figure 2. The sources of information are MS Excel files (with thousands of rows) and a Visual Paradigm project. A custom preprocessor written in `C#`, is used to parse the MS Excel files applying some heuristics to generate the PVSS datapoints names and the corresponding aliases. Then based on the aliases it generates a tree structure following the Product Breakdown Structure (PBS) of the detector. However those inputs cannot be easily compared with a `diff` tool either. The TOTEM DCS has a centralized definition for all the datapoints settings related to archiving, alarms, units and screen representations parameters. A dedicated procedure enforces that all the datapoints match certain filters and conform to central configuration standards.

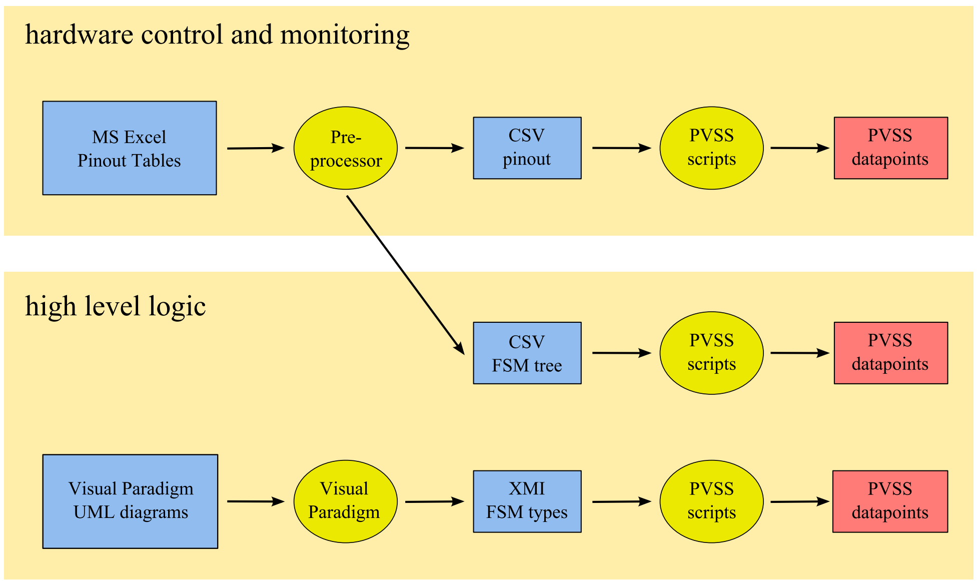


Figure 2. Usage of automation tools within the TOTEM DCS

Therefore neither using the `.dpl` files to deploy the system, nor the existing TOTEM tools, have an advantage when it comes to compare the final systems.

One requirement for the deployment of our code into the computers managed by other teams, was to wrap it up into packages following the JCOP guidelines. This, together with the fact of having three independent detectors, and dividing the code by functions such as High Voltage, Low Voltage, Environmental Sensors results in a sizable number of components based on the common ones that provide the automation features.

The problem is that small changes in the automation code, or changes in the filters that define the archiving parameters, units, alarms can produce undesired side effects in other areas or other TOTEM detectors. Even small changes in part of the code might have unexpected effects in the run-time behavior of the project.

The procedure that the regression tool uses for comparing system is based on building a new PVSS project from scratch, and installing all the JCOP and TOTEM components. It follows the procedures described in the internal document for ‘new TOTEM DCS developers’.

Providing a detailed list of the steps, they are:

- Create an empty project.
- Install the JCOP framework installation tool.
- Install the desired JCOP framework components and TOTEM components.
- Export the datapoints using a call to the `pvss00ascii.exe` executable.
- Clean up the resulting `.dpl` file in order to get rid of unwanted / nonstructural information.
- Compare the cleaned `.dpl` file with a clean `.dpl` reference file that has been peer validated in the team of developers and by production users.
- Create a user friendly report out of the information gathered and send it to relevant parties.

It is important to make clear that PVSS is able to import partial `.dpl` files. By removing the timestamp columns or some other columns such as the internal `DpId` the `.dpl` file is still valid, and can be imported into a PVSS system.

3. Related work

The JCOP Framework team in the EN-ICE group at CERN have also been investing in Quality Assurance. They perform regular automated regression testing of the JCOP Framework at the User Interface level [7]. They plan unit testing for the near future. EN-ICE and TOTEM have adopted Python as the language for their test suites for its clean interface, extensive library of utilities and cross-platform compatibility. In this way both groups have been able to profit from a mutual exchange of software. Examples include a PVSS `PMON` communication wrapper written in Python (enabling the monitoring and control of all processes in a PVSS project) and the 'castration tool' described in the next section.

Note that in TOTEM, we are not attempting to test the JCOP Framework. Neither are we attempting to test PVSS nor the Operating System itself. Rather, in TOTEM, we assume the correctness of these underlying systems. Our primary aim is to maintain and validate the reliability and consistency of our back-end systems in the face of changes in our initial requirements. For this reason, we have not undertaken any User Interface testing at this point.

4. Implementation

4.1. Overview of the files

Figure 3 shows all the Python files that form the tool and their dependencies. There are 3 additional config files in XML format: `autoTestCastrateConfig.xml`, `autoTestConfig.xml` and `autoTestPvssLogProcessorConfig.xml`.

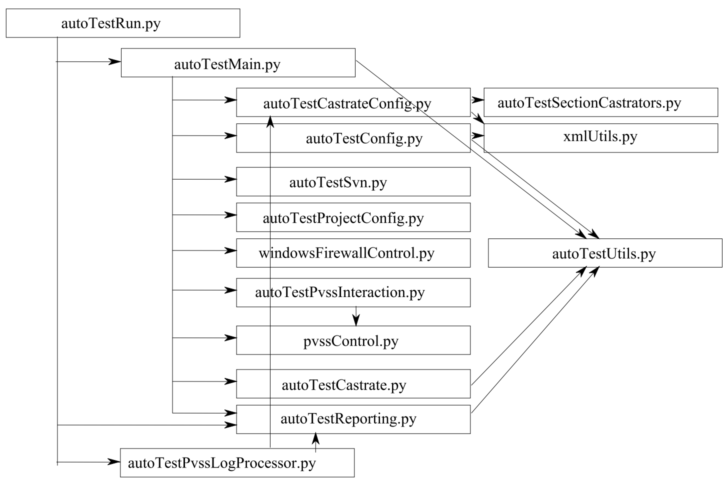


Figure 3. Python files dependencies

Additional files provide support to schedule the tool for running at night, and execute it with adequate privileges.

4.2. Formalization of the steps

- Kill any PVSS related process running in the machine.
- Export all the components stored in SVN repositories as defined in the config files.
- Export from SVN the reference `.dpl` file.
- Configure the Windows firewall for all the PVSS projects.
- Create a PVSS project. We invoke the standard PVSS procedure from an special User Panel not linked to any particular project. Currently this panel is stored in the component `totInstallation`.

- Modify the generated project to include the `fwInstallation` and `totInstallation` in the `proj_path` variable of its `config` file.
- Modify the generated project `config` file to include all the components specified in the config file (and exported from SVN) into a `totInstallation` section.
- Start the project.
- Append and additional manager from `totInstallation` into the project using the `PMON` TCP protocol.
- Wait that `totInstallation` has finished installing all the desired components. The tool monitors the custom manager status and the global project status using the `PMON` TCP protocol. There can be several restarts of the project during this process.
- Stop the project using the `PMON` TCP protocol.
- Merge PVSS logs and provide a resume of the errors during the execution; save them to the hard disk.
- Export the datapoints into a `.dpl` file.
- Commits the `.dpl` file into SVN.
- Clean up the resulting `.dpl` file.
- Clean up the reference `.dpl` file.
- Compare both `.dpl` files and generate a temporary report in the hard disk.
- Send by email the report and PVSS logs.

4.3. Details on the project config file manipulations

The project `config` file is updated before the start up of the project. The first step is to introduce new search paths:

```
proj_path = "D:/autotesting/export/fwInstallation_6.0.11"
proj_path = "D:/autotesting/export/totInstallation"
proj_path = "D:/autotesting/project/fwComponents"
proj_path = "D:/autotesting/project"
```

Later in the file we list all the desired components and where to find them. This file also accepts additional sections identified by `[sectionname]` followed by several entries of the form `key=value`. A key can be repeated several times, and when it is read by PVSS it will result in an array.

```
[totem]
DestinationComponents = "D:/autotesting/project/fwComponents"
InstallComponents = "D:/autotesting/export/framework_4.3.0/fwCore.xml"
InstallComponents = "D:/autotesting/export/framework_4.3.0/fwAnalogDigital.xml"
InstallComponents = "D:/autotesting/export/framework_4.3.0/fwCaen.xml"
InstallComponents = "D:/autotesting/export/framework_4.3.0/fwTrending.xml"
...
InstallComponents = "D:/autotesting/export/fwFsm_28.7.1/fwFSM.xml"
InstallComponents = "D:/autotesting/export/fwElmb_4.2.2/fwElmb.xml"
...
InstallComponents = "D:/autotesting/export/totServices/totServices.xml"
InstallComponents = "D:/autotesting/export/totRadmon/totRadmon.xml"
InstallComponents = "D:/autotesting/export/totFsmTypes/totFsmTypes.xml"
InstallComponents = "D:/autotesting/export/totAutomation/totAutomation.xml"
InstallComponents = "D:/autotesting/export/totHelp/totHelp.xml"
...
```

4.4. totInstallation

The basic JCOP `fwInstallation` tool (that also fulfils the JCOP component packaging rules) is not focused on automating the installation of a project without user intervention. To overcome these limitations, `totInstallation` takes over the responsibility of deciding when and how to install components and internally calls `fwInstallation`. The component `totInstallation` is merely a wrapper adding extra functionally.

A key step for the process is a panel able to create a project just executing a command line:

```
PVSS00ui.exe -n +config config.pa -silentMode -menuBar -iconBar -centered \
    -p "D:\...\CreateProject.pnl",$projectName:"project",$installDir:"D:\autotesting"
```

The invocation of panels allow to pass command line arguments, however the invocation of control managers do not allow any additional parameter. This kind of invocation with the `-n` do not need any previous project in the system, as there is no connection to any PVSS event or data manager.

Additionally `totInstallation` reads all the entries in the `[totem]` section of the project `config` file. It verifies if the components are installed, and forces their installation one by one, making sure that the `postInstall` scripts are properly executed and finished before advancing to the next one. Although it could be easy to perform restarts of the projects where the component demand it, the scripts do not perform any restarts for the project. At the present time the only component that really needs a restart is `fwFSM`, and the installation of `totServices` determines the proper conditions and executes the restart. By avoiding unnecessary (very lengthy) restarts we are able to reduce the overall execution time.

4.5. Cleanup of .dpl files: The castrator

The castration tool is derived from the work of EN-ICE, but with some additions and better interfacing to our environment.

The datapoint `.dpl` files are structured in sections starting with a # and a section name. Following this line a tab separated table with column names in the first line is the content of the section. The quotation character is #, also enabling fields to reach over several lines (hence the fields are allowed to contain new line characters if quoted). However the section called `DpType` follows a different syntax. They are the datapoints `type` definition and it is left unchanged.

Castration of the datapoint list means to preserve only the structural information of these datapoints and not their value contents. Castration strips columns and rows containing only nonstructural information. Each distinct section of the datapoint file has to be inspected to determine the right columns to be copied. Typically uninteresting columns are:

- Timestamps like `StampSec`, `StampMSec` and `_original.._stime`. All occurrences of this are striped from sections containing them, such as `DpValue`, `PvssRangeCheck`, `DpDefaultValue`, `AlertClass`, `AlertValu`, `DistributionInfo`, `DpFunction`, `DpConvRawToIngMain`, `DpConvIngToRawMain`, `DpSmoothMain`, `PeriphAddrMain`, `DbArchiveInfo`
- Datapoint value like `_original.._value` (they only occur in the `DpValue` section).
- Datapoint status like `_original.._status64` (they only occur in the `DpValue` section).
- Datapoint internal identifiers like `DpId`, which only reflects the order in which the datapoints were created. Since the installation process is taking place in a distributed manner over several PVSS managers, small timing differences may already lead to a different numbering. (they only occur in the `Datapoint/DpId` section).

First of all the castrate tool separates a single `.dpl` into the different sections. Then by using the filters of the tool .xml config files defines what columns are of interest, by explicitly selecting them with the `<columnsToCopy>` XML elements. As there is no guarantee of the sorting of the lines within the section, we introduce the `index` tag within the `section` definition. This index is useful for sorting the lines before the comparison steps, and therefore ignoring differences related to different ordering. Also the `<index>` relates with the `<filterIgnore>` elements, when those of those filters is ignored, the full line is ignored. There are some computer specific datapoints generated by PVSS.

An example of this castration definition from `autoTestCastrateConfig.xml` is given:

```
<castrateConfig>
    <section index="ElementName" name="DpValue">
        <filterIgnore>
          <match regexp="^_Stat_event_\d_to_ctrl_\d\..*"/>
          <match regexp="^_Stat_event_\d_to_event_\d\..*"/>
          <match regexp="^_Stat_event_\d_to_dist_\d\..*"/>
          <match regexp="^_Stat_event_\d\..*"/>
          <match regexp="^_DistManager\..*"/>
          <match regexp="^_DistConnections\..*"/>
          <match regexp="^_ArchiveDisk\..*"/>
          <match regexp="^_ValueArchive_\d\..*"/>
          <match regexp="^_Event.License\..*"/>
          <match regexp="^_mp_COUNTER1\..*"/>
          <match regexp="^_unDistributedControl_dist_\d\..*"/>
        </filterIgnore>
        <columnsToCopy>
            <!--Note : the index 'ElementName' is copied as well-->
            <column name="TypeName"/>
            <!--<column name="_original.._value"/>-->
            <!--<column name="_original.._status64"/>-->
            <!--<column name="_original.._stime"/>-->
        </columnsToCopy>
    </section>
</castrateConfig>
```

4.6. PVSS logs filtering

The TOTEM regression testing tool permits filtering out the error messages that may be safely ignored.

Only the `SEVERE`, `FATAL` and `WARNING` error messages that not explicitly suppressed are included in the HTML body of the final report.

An example of log message filtering from `autoTestPvssLogProcessorConfig.xml` is:

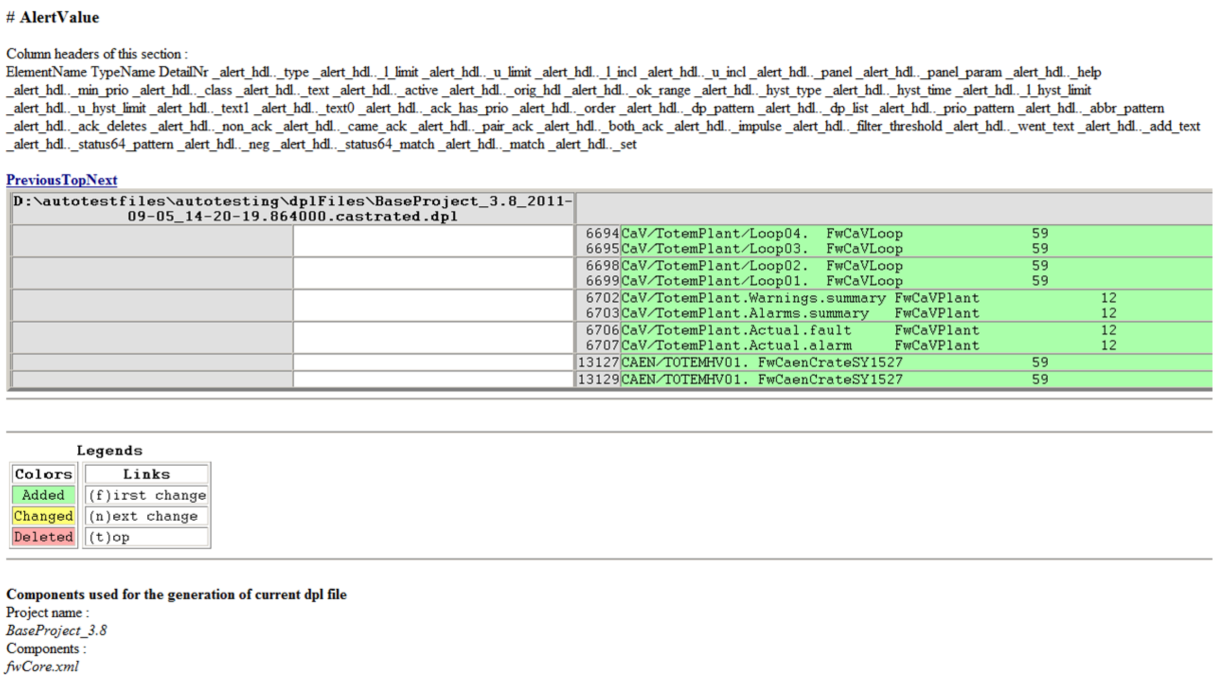
```
<pvssLogFilterIgnore>
<match regexp="^Blocking Manager .* detected\. No heartbeat since 30 seconds\."/>
<match regexp="^Manager .* is no longer blocking\."/>
<match regexp="^Unexpected state, RemoveItem The value of the handle is invalid"/>
<match regexp="^Unexpected state, RemoveItem The operation completed successfully\."/>
<match regexp="^Unexpected state, statFunc work: , .*, Omitted .* periods in calculation"/>
<match regexp="^Values were discarded, .*, .*, function has 200 pending runs -> DISCARDING!"/>
<match regexp="^Not processing writes now"/>
...
</pvssLogFilterIgnore>
```

5. Reporting

At the end of each daily execution, the tool sends an email using a mailing list to the TOTEM collaborators that want to follow closely the changes. Typically it is the development team who subscribe to the notifications.

Changes between the `.dpl` files are compared using the Python `difflib` libraries and included in the report. On the left side it shows the validated file, and on the right side any modifications. Python `difflib` uses a color encoding to clarify if they are additions, removals, or changes. An example is give in Figure 4.

At the end of the report there is a list with the components used, the filtered error messages, and information about the running time and OS platform used for the test. The full PVSS log of the installation process is emailed as a compressed attachment.



# AlertValue

Column headers of this section :
ElementName TypeName DetailNr _alert_hdl.._type _alert_hdl.._l_limit _alert_hdl.._u_limit _alert_hdl.._l_incl _alert_hdl.._u_incl _alert_hdl.._panel _alert_hdl.._panel_param _alert_hdl.._help _alert_hdl.._min_prio _alert_hdl.._class _alert_hdl.._text _alert_hdl.._active _alert_hdl.._orig_hdl _alert_hdl.._ok_range _alert_hdl.._hyst_type _alert_hdl.._hyst_time _alert_hdl.._l_hyst_limit _alert_hdl.._u_hyst_limit _alert_hdl.._text1 _alert_hdl.._text0 _alert_hdl.._ack_has_prio _alert_hdl.._order _alert_hdl.._dp_pattern _alert_hdl.._dp_list _alert_hdl.._prio_pattern _alert_hdl.._abbr_pattern _alert_hdl.._ack_deletes _alert_hdl.._non_ack _alert_hdl.._came_ack _alert_hdl.._pair_ack _alert_hdl.._both_ack _alert_hdl.._impulse _alert_hdl.._filter_threshold _alert_hdl.._went_text _alert_hdl.._add_text _alert_hdl.._status64_pattern _alert_hdl.._neg _alert_hdl.._status64_match _alert_hdl.._match _alert_hdl.._set

PreviousTopNext

D:\autotestfiles\autotesting\dplFiles\BaseProject_3.8_2011-09-05_14-20-19.864000.castrated.dpl		
		6694 CaV/TotemPlant/Loop04. FwCaVLoop 59 6695 CaV/TotemPlant/Loop03. FwCaVLoop 59
		6698 CaV/TotemPlant/Loop02. FwCaVLoop 59 6699 CaV/TotemPlant/Loop01. FwCaVLoop 59
		6702 CaV/TotemPlant.Warnings.summary FwCaVPlant 12 6703 CaV/TotemPlant.Alarms.summary FwCaVPlant 12
		6706 CaV/TotemPlant.Actual.fault FwCaVPlant 12 6707 CaV/TotemPlant.Actual.alarm FwCaVPlant 12
		13127 CAEN/TOTEMHV01. FwCaenCrateSY1527 59
		13129 CAEN/TOTEMHV01. FwCaenCrateSY1527 59

Legends

Colors	Links
🟩 Added	(f)irst change
🟨 Changed	(n)ext change
🟥 Deleted	(t)op

Components used for the generation of current dpl file
Project name :
BaseProject_3.8
Components :
fwCore.xml

Figure 4. Example of file differences as reported by email

6. Possible improvements

Keeping in mind that the approach is to test all our development and data points generation steps as a black box, there are still many possible improvements in this project.

Some of them are:

- Do a comparison not only of `.dpl` files, but also of the directory structure. Some files are created dynamically during project execution, other files are copied according to the platform, and it is easy to forget to remove or include new files in the components definition for the JCOP installation process.
- Partial extraction of `.dpl` files and their comparison could help in unit testing.
- Extend the regression testing to cover User Interface aspects.
- Provide better feedback about the time used for every component installation. Some of them take minutes, while others can take more than one hour. Sometimes changes can dramatically affect the installation time, without compromising the execution time of controlling and monitoring the hardware.
- The Python comparison library uses too much memory (>2 GB). A less memory demanding library would be ideal. It could be a custom tool based on the `.dpl` structure.
- Monitor CPU usage.
- Mailing lists do not accept mails greater than 10 MB. Frequently major changes and reconfigurations produce a diff report bigger than this. As an indicator, the `.dpl` plain text file representing the whole TOTEM DCS is close to 30 MB. A small change in the archiving settings or the alarm limits can easily generate changes in hundreds of datapoints. The resulting report showing the differences is encoded as HTML instead as plain text and consequently this 10 MB limit becomes a problem.
- Improve the JCOP `fwInstallation` tool to have a local back-end of desired PVSS components for the project, and merge it with `totInstallation`. Although implemented, this synchronization mechanism for installing components needs a remote database.

7. Conclusions

The TOTEM regression testing tool can be used not only for rapidly verifying small improvements but can also help in assessing changes between major releases.

TOTEM operates with brief data taking periods. This allows the DCS team much flexibility for deploying updates. We are able to focus mainly on small and incremental changes ensuring reliability. Right now we can send an update, and if it fails, we have enough time before the next run to fix any issue. In a longer term, in a continuous running scenario we could easily adapt by extracting `.dpl` files from the production systems and verifying using a system generated in a development environment. Regression testing and comparison applies not only across different releases but also among environments (production and development).

The tool itself does not provide full code coverage testing and has many limitations. However if after introducing changes, the `.dpl` file evolves as expected and there are no additional error messages the developers are extremely confident that an upgrade will not produce any side effects, (or at least it will not be any worse that it was before!).

Still the process takes too long, about 5 hours. It would be ideal if it were less than 1 hour opening the doors to a much more agile development life-cycle, and using it as a kind of compiler to confirm the changes several times per day. The TOTEM DCS team has tried to reduce this time as much as possible, however they have been unable to optimize more the regenerating process for a complete system.

Finally if a JCOP component alone is changed without changing our TOTEM components, we can even identify bugs in the JCOP framework. We are able to explore the compatibility with newer component versions before the production systems are upgraded. In this way we have discovered several bugs in the JCOP framework and reported them. Our test procedure is a huge use case where we not only test our code, but indirectly all the JCOP and PVSS layers. We could also validate different PVSS versions and PVSS patches using the same procedure.

References

[1] Albrow M et al. 2006 *Prospects for Diffractive and Forward Physics at the LHC* CERN/LHCC 2006-039/G-124.

[2] Anelli G et al. 2008 The TOTEM experiment at the CERN Large Hadron Collider *Journal of Instrumentation* 3.

[3] Antchev G et al. 2011 First measurement of the total proton-proton cross-section at the lhc energy of sqrt(s) = 7 tev *A letters journal exploring the frontiers of physics* 96.

[4] Antchev G et al. 2011 Proton-proton elastic scattering at the lhc energy of sqrt(s) = 7 tev *A letters journal exploring the frontiers of physics* 95.

[5] Atanassov I, Lucas Rodríguez F, Palazzi P, Ravotti F, Stöckell S, Sziklai J, and Tulimaki V 2010 Automation tools in the software development of the totem detector control system *2010 IEEE Nuclear Science Symposium Conference Record* (On behalf of the TOTEM Collaboration).

[6] Berardi V et al. 2004 *TOTEM: Technical Design Report* CERN-LHCC-2004-002.

[7] Burkimsher PC, González Berges M, and Klikovits S 2011 Multi-platform scada gui regression testing at cern *Proceedings of ICALEPCS* Grenoble, France.

[8] ETM A.G. 2008 *ProzessVisualisierungs und SteuerungsSystem (PVSS)*.

[9] JCOP Framework Team 2007 *Joint COntrols Project (JCOP) framework subproject: guidelines and conventions* CERN-JCOP-2000-008.

[10] Lucas Rodríguez F 2010 Estimation of the response time and data flows in the totem detector control system *Proceedings of the Eighth International Workshop on Personal Computers and Particle Accelerators*.

[11] Lucas Rodríguez F, Atanassov I, Palazzi P, and Ravotti F 2009 The totem detector control system *Proceedings of the 12th International Conference On Accelerator And Large Experimental Physics Control Systems* (On behalf of the TOTEM Collaboration).

[12] OPC Foundation 1996 *OLE for Process Control*.