Geant-val:
a web application for validation of detector simulations

Luc Freyermuth[1], *Dmitri* Konstantinov[2,3,*], *Grigorii* Latyshev[2,**],
Ivan Razumov[2,3], *Witold* Pokorski[3]and *Alberto* Ribon[3]

[1]EISTI (Cergy, France)
[2]NRC "Kurchatov Institute" – IHEP (Protvino, Russia)
[3]CERN

Abstract. One of the key factors for the successful development of Monte-Carlo programs for physics simulations is to properly organize regression testing and validation. Geant4, the world-standard toolkit for HEP detector simulation, heavily relies on this activity. The CERN SFT group, which contributes to the development, testing, deployment and support of the toolkit, is also in charge of running on a monthly basis a set of community-developed tests using the development releases of Geant4. We present the web application Geant-val developed for visualizing the results of these tests so that comparisons between different Geant4 releases can be made. The application is written using Express.js, Node.js and Angular frameworks, and uses PostgreSQL for storing test results. Test results are visualised using ROOT and JSROOT. In addition to pure visual comparisons, we perform different statistical tests (χ^2, Kolmogorov-Smirnov, etc) on the client side using JavaScript Web Workers.

1 Introduction

Geant4 [1] is a toolkit for the simulation of the passage of particles through matter. Its areas of application include high energy, nuclear and accelerator physics, as well as studies in medical and space science.

The Geant4 team produces a public release once a year, with monthly internal testing releases. These releases need to be validated.

Physics validation of Geant4, which includes regression testing (comparing physics output from one release to another), comparison of performance between different physics models and validation testing (comparing physics output with published experimental results), requires analysing a large number of data produced by Geant4 tests.

Geant4 validation procedure consists of two steps: generating results (executing the tests and importing their output into Geant-val) and analysing these results.

*e-mail: Dmitri.Konstantinov@cern.ch
**e-mail: Grigorii.Latyshev@cern.ch

2 Requirements

A validation of software is a complex process involving entire Geant4 community starting from developers of core functionality end to users. Below we briefly list the requirements of validation system:

- Simple integration of current set of Geant4 tests;
- Support for various test's output formats – ROOT and plain text;
- **High parallelism** is needed to be possible run a large set of Geant4 tests in reasonable time. To achieve this the system should provide a way to run Geant4 tests in highly diversificated jobs in batch systems.
- Frienly user interface;
- User-driven developing;

3 Components of validation system

The components of Geant-val web application and interactions between them are shown in the Fig. 1.

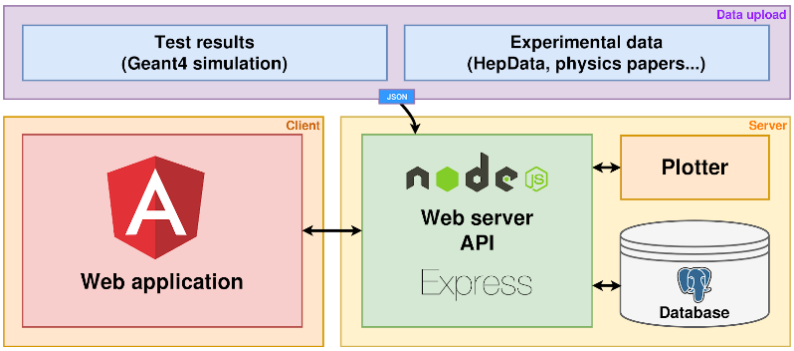


Figure 1. Components of Geant-val web application and flow of data between them.

The server is the core of the Geant4 validation system. It provides a web API that allows clients to access the database, respond to the clients' requests and generate high quality plots "on the fly" whenever they are requested using the ROOT [4] data analysis framework. It is written in JavaScript and runs with the Node.js engine [5].

The database is used for storing plots containing simulation results or experimental data, together with parameters describing these plots: tool name and version (e.g. `Geant4, 10.4.p02`), name of the test by which the plot was produced, or Inspire (HepData) ID of the article, etc. PostgreSQL [6] is used as database management system for the application. The database instance is provided by the CERN Database-On-Demand service

A ROOT-based C++ plotting utility was developed to produce high quality plots. It uses data in the JSON format (see Appendix 1) which has been introduced as main interchange format between all parts of the application. The utility is not deeply integrated in the server infrastructure and can be used as a standalone application. It supports all types of application's data, can plot histograms with different binning on one canvas, and produce ratio plots. Ranges and scales of plot axes are selected automatically, with ability to override if necessary.

Web interface is user-friendly AngularJS [7] single page application which provides plots with tests results together with statistical analysis to the users.

Each plot on the website can be displayed as a static image produced by `plotter` or as an interactive JSROOT [8] object, which allows changing axes ranges, scales and styles of the data "on the fly" in the same way as it can be done in ROOT. It is possible to export the plots in PNG, ROOT, EPS, Gnuplot [9] or Geant-val JSON format.

Access to the test results produced with internal Geant4 releases is restricted to the Geant4 developers authenticated via CERN Single Sign-On. For public Geant4 releases the access is open to anyone.

To manage a set of Geant4 tests and their configurations, a `mc-config-generator` **framework** was developed. It allows one to configure and run test jobs in various batch systems (CERN LSF, HTCondor, Torque PBS), and to convert the results into JSON format for further uploading to the application database. The framework is not Geant4-specific, and can be used with other projects (e.g., Pythia8). Source code is available in corresponding Git repository[1].

4 User interface

The Geant-val website provides two ways of viewing and comparing results:

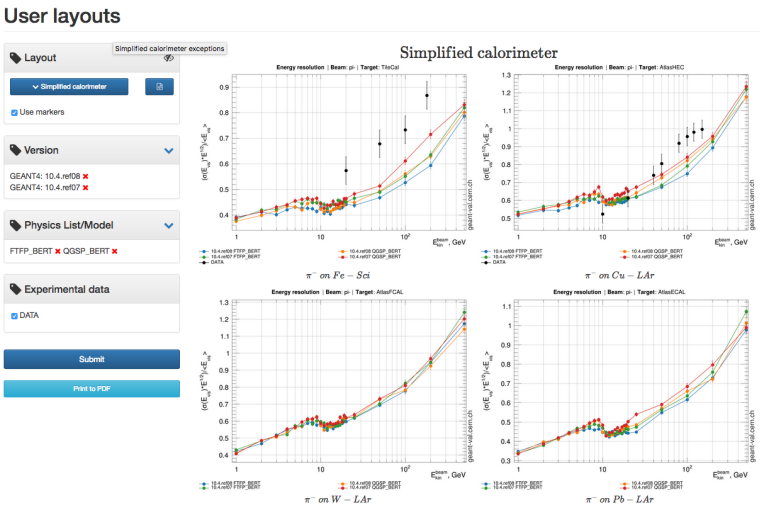
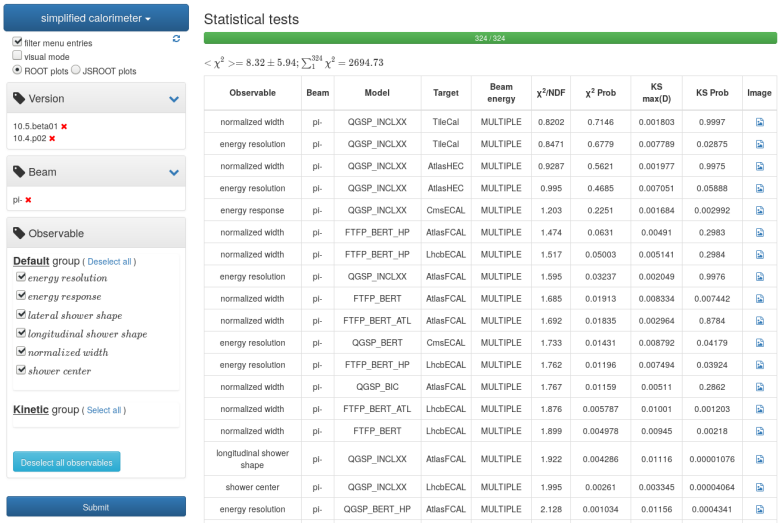


Figure 2. Example of user defined layout for the Geant4 "simplified calorimeter" test showing test results for two Geant4 reference releases - 10.4.ref07 and 10.4.ref08.

Statistical comparisons page (see Fig. 3) allows one to perform comparison of simulation with compatible experimental results using statistical tests. Currently χ^2 ($\chi^2/n.d.f.$, χ^2 probability) and Kolmogorov-Smirnov (KS Max(D), KS probability) tests are implemented. All computations are performed asynchronously on the client side using JavaScript WebWorkers. For this purpose, JavaScript code to perform χ^2 and Kolmogorov-Smirnov tests have been written, and their results cross-checked against the same statistical techniques implemented in the ROOT framework.

[1] https://gitlab.cern.ch/GeantValidation/geant-config-generator



Observable	Beam	Model	Target	Beam energy	χ^2/NDF	χ^2 Prob	KS max(D)	KS Prob	Image
normalized width	pi-	QGSP_INCLXX	TileCal	MULTIPLE	0.8202	0.7146	0.001803	0.9997	
energy resolution	pi-	QGSP_INCLXX	TileCal	MULTIPLE	0.8471	0.6779	0.007789	0.02875	
normalized width	pi-	QGSP_INCLXX	AtlasHEC	MULTIPLE	0.9287	0.5621	0.001977	0.9975	
energy resolution	pi-	QGSP_INCLXX	AtlasHEC	MULTIPLE	0.995	0.4685	0.007051	0.05888	
energy response	pi-	QGSP_INCLXX	CmsECAL	MULTIPLE	1.203	0.2251	0.001684	0.002992	
normalized width	pi-	FTFP_BERT_HP	AtlasFCAL	MULTIPLE	1.474	0.0631	0.00491	0.2983	
normalized width	pi-	FTFP_BERT_HP	LhcbECAL	MULTIPLE	1.517	0.05003	0.005141	0.2984	
energy resolution	pi-	QGSP_INCLXX	AtlasFCAL	MULTIPLE	1.595	0.03237	0.002049	0.9976	
normalized width	pi-	FTFP_BERT	AtlasFCAL	MULTIPLE	1.685	0.01913	0.008334	0.007442	
normalized width	pi-	FTFP_BERT_ATL	AtlasFCAL	MULTIPLE	1.692	0.01835	0.002964	0.8784	
energy resolution	pi-	QGSP_BERT	CmsECAL	MULTIPLE	1.733	0.01431	0.008792	0.04179	
energy resolution	pi-	FTFP_BERT_HP	LhcbECAL	MULTIPLE	1.762	0.01196	0.007494	0.03924	
normalized width	pi-	QGSP_BIC	AtlasFCAL	MULTIPLE	1.767	0.01159	0.00511	0.2862	
normalized width	pi-	FTFP_BERT_ATL	LhcbECAL	MULTIPLE	1.876	0.005787	0.01001	0.001203	
normalized width	pi-	FTFP_BERT	LhcbECAL	MULTIPLE	1.899	0.004978	0.00945	0.00218	
longitudinal shower shape	pi-	QGSP_INCLXX	AtlasFCAL	MULTIPLE	1.922	0.004286	0.01116	0.00001076	
shower center	pi-	QGSP_INCLXX	LhcbECAL	MULTIPLE	1.995	0.00261	0.003345	0.00004064	
energy resolution	pi-	QGSP_BERT_HP	AtlasFCAL	MULTIPLE	2.128	0.001034	0.01156	0.0004341	

Figure 3. Example of statistical comparison between two official Geant4 releases, 10.5.beta01 and 10.4.p02, for the "simplified calorimeter" test.

User layouts page (see Fig. 2). Some Geant4 tests produce hundreds of different plots, but for fast "visual" validation it is often enough to compare only a small well-defined subset of them. The *User layout* is an XML file describing what plots should be displayed and how should they be laid out on a page. User can use one of the existing layouts or define their own one (see Appendix 2).

5 Conclusion

The presented software provides an uniform and clear way of performing all stages of physics validation: from creating configuration files and running simulation code to producing final plots.

The Geant-val web application and the `mc-config-generator` framework are actively used by Geant4 developers in CERN and by Geant4 Medical Simulation Benchmark Group (G4BSMG).

References

[1] S. Agostinelli *et al.* [GEANT4 Collaboration], Nucl. Instrum. Meth. A **506**, 250 (2003). doi:10.1016/S0168-9002(03)01368-8

[2] INSPIRE-HEP: High-Energy Physics Literature Database http://inspirehep.net/

[3] E. Maguire, L. Heinrich and G. Watt, J. Phys. Conf. Ser. **898** (2017) no.10, 102006 doi:10.1088/1742-6596/898/10/102006 [arXiv:1704.05473 [hep-ex]].

[4] Rene Brun and Fons Rademakers, ROOT - An Object Oriented Data Analysis Framework, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also (http://root.cern.ch).

[5] NodeJS is a JavaScript runtime https://nodejs.org/

[6] PostgreSQL is an open-source database https://www.postgresql.org/

[7] AngularJS is an MWM JavaScript framework https://angularjs.org/
[8] Bertrand Bellenot and Sergey Linev 2015, J. Phys.: Conf. Ser.,664, 062033
[9] Gnuplot is a portable command-line driven graphing utility http://www.gnuplot.info/

6 Appendix

```
{
  "article": {"inspireId": -1},
  "mctool": {"name": "Geant4", "version": "", "model": ""},
  "testName": "",
  "metadata": {"observableName": "", "reaction": "",
    "targetName": "", "beamParticle": "",
    "beamEnergies": [], "secondaryParticle": "",
    "parameters": [
      {
        "names": "THETA",
        "values": "60 degrees"
      }
    ]
  },
  // for scatter data
  "plotType": "SCATTER2D",
  "chart": {
    "nPoints: 0,
    "title": "", "xAxisName": "", "yAxisName": "",
    "xValues": [], "yValues": [],
    "xStatErrorsPlus": [], "xStatErrorsMinus": [],
    "yStatErrorsPlus": [], "yStatErrorsMinus": [],
    "xSysErrorsPlus": [], "xSysErrorsMinus": [],
    "ySysErrorsPlus": [], "ySysErrorsMinus": []
  },
  // for histogram data
  "plotType": "TH1",
  "histogram": {
    "nBins: [0],
    "title: "", "xAxisName": "", "yAxisName": "",
    "binEdgeLow": [], "binEdgeHigh": [], "binContent": [],
    "yStatErrorsPlus": [], "yStatErrorsMinus": [],
    "ySysErrorsPlus": [], "ySysErrorsMinus": [],
    "binLabel": []
  }
}
```

Appendix 1. Example of JSON format used by the web application.

```
<?xml version="1.0" encoding="UTF-8"?>
<layout>
  <default model="FTFP_BERT"></default>
  <row>
    <label text="\LARGE{tileatlas}" colspan="4"/>
  </row>
  <!-- energy response, normalized width, longitudinal shower shape and
      ↪ lateral shower shape -->
  <row>
    <plot test="tileatlas" observable="relative␣energy␣deposition" beam="e-"
        ↪ energy="10"   secondary="None" target="tileatlas" parname="THETA"
        ↪  parvalue="0␣degrees" yaxis="lin" xaxis="lin" title=""/>
    <plot test="tileatlas" observable="relative␣energy␣deposition" beam="e-"
        ↪ energy="10"   secondary="None" target="tileatlas" parname="THETA"
        ↪  parvalue="20␣degrees" yaxis="lin" xaxis="lin" title=""/>
  </row>
  <row>
    <plot test="tileatlas" observable="relative␣energy␣deposition" beam="e-"
        ↪ energy="20"   secondary="None" target="tileatlas" parname="THETA"
        ↪  parvalue="21␣degrees" yaxis="lin" xaxis="lin" title=""/>
    <plot test="tileatlas" observable="relative␣energy␣deposition" beam="e-"
        ↪ energy="20"   secondary="None" target="tileatlas" parname="THETA"
        ↪  parvalue="61␣degrees" yaxis="lin" xaxis="lin" title=""/>
   </row>
  <row>
    <plot test="tileatlas" observable="relative␣energy␣deposition" beam="e-"
        ↪ energy="100"   secondary="None" target="tileatlas" parname="THETA
        ↪ " parvalue="0␣degrees" yaxis="lin" xaxis="lin" title=""/>
    <plot test="tileatlas" observable="relative␣energy␣deposition" beam="e-"
        ↪ energy="100"   secondary="None" target="tileatlas" parname="THETA
        ↪ " parvalue="10␣degrees" yaxis="lin" xaxis="lin" title=""/>
    <plot test="tileatlas" observable="relative␣energy␣deposition" beam="e-"
        ↪ energy="100"   secondary="None" target="tileatlas" parname="THETA
        ↪ " parvalue="20␣degrees" yaxis="lin" xaxis="lin" title=""/>
  </row>
</layout>
```

Appendix 2. Example of User layout XML file.