arXiv:physics/0605063 v1 7 May 2006

Concepts, Developments and Advanced Applications of the PAX Toolkit

S. Kappler*, M. Erdmann, M. Kirsch, G. Müller (RWTH Aachen university, Germany),
J. Weng (CERN, Geneva, Switzerland), A. Floßdorf (DESY, Hamburg, Germany),
U. Felzmann, G. Quast, C. Saout, A. Schmidt (Karlsruhe university, Germany)

Abstract

The Physics Analysis eXpert (PAX) is an open source toolkit for high energy physics analysis. The C++ class collection provided by PAX is deployed in a number of analyses with complex event topologies at Tevatron and LHC. In this article, we summarize basic concepts and class structure of the PAX kernel. We report about the most recent developments of the kernel and introduce two new PAX accessories. The PaxFactory, that provides a class collection to facilitate event hypothesis evolution, and VisualPax, a Graphical User Interface for PAX objects.

INTRODUCTION

Physics analyses at modern collider experiments enter a new dimension of event complexity. At the LHC, for instance, physics events will consist of the final state products of the order of 20 simultaneous collisions. In addition, a number of todays physics questions is studied in channels with complex event topologies and configuration ambiguities occurring during event analysis.

The Physics Analysis eXpert toolkit (PAX) is a continuously maintained and advanced C++ class collection, specially designed to assist physicists in the analysis of complex scattering processes [1, 2, 3, 4]. PAX is realized in the C++ programming language [5]. It provides additional functionality in top of the vector algebra of the widely-spread libraries CLHEP [6] (default) or ROOT [7]. The PAX container model as well as file I/O are based on the C++ Standard Template Library (STL) [5].

THE PAX KERNEL

The class collection of the PAX kernel allows the definition of an abstraction layer beyond detector reconstruction by providing a generalized, persistent HEP event container with three types of physics objects (particles, vertices and collisions), relation management and file I/O scheme. The PAX event container is capable of storing the complete information of multi-collision events (including decay trees with spatial vertex information, four-momenta as well as additional reconstruction data). An automated copy functionality for the event container allows the analyst to consistently duplicate event containers for hypothesis evolution, including its physics objects and relations. PAX physics objects can hold pointers to an arbitrary number of instances of arbitrary C++ classes, allowing the analyst to keep track of the data origin within the detector reconstruction software. Further advantages arising from the usage of the PAX toolkit are a unified data model and nomenclature, and therefore increased code lucidity and more efficient team work. The application of the generalized event container provides desirable side-effects, such as protection of the physics analysis code from changes in the underlying software packages and avoidance of code duplication by the possibility of applying the same analysis code to various levels of input data.

PAX physics objects

The three types of generalized physics objects provided by the PAX kernel are:

- Particles (or reconstructed objects), i.e. Lorentz-vectors, are represented by the class *PaxFourVector*.
- Vertices, i.e. three-vectors, represented by the class *PaxVertex*, are foreseen to realize particle decays.
- Collisions, represented by the class *PaxCollision*, are foreseen to allow the separation of multiple interactions in high-luminostity environments.

The vector characteristics of the classes *PaxFourVector* and *PaxVertex* is inherited from the corresponding classes of the CLHEP or ROOT libraries. Commonly needed, additional properties such as a name, particle-id, status, charge, a workflag etc. can be stored in data members. Specific information complementary to data members, such as b-tags, jet cone sizes or energy corrections, for instance, can be stored in the so-called user records (i.e. collections of string-double pairs).

Each PAX physics object can record pointers to an arbitrary number of instances of arbitrary C++ classes. This way, the user can keep track of the data origin within the detector reconstruction software, for instance. Access to the pointers is possible at the same runtime during any later stage of the analysis.

Copy constructors are provided to perform deep copies of PAX physics objects. When copying a PAX physics object, all pointers are copied as well. For the convenience of reconstructing particle decay chains, PAX physics objects are enabled to establish relations and can be organized in containers based on the STL class templates *map<key, item>* and *multimap<key, item>*, respectively.

* Corresponding author: Steffen.Kappler@cern.ch

Event container

PAX provides a generalized event container for storage and handling of the complete information of one multicollision event including decay trees, spatial vertex information, four-momenta as well as additional reconstruction data in the user records.

This container is represented by the class *PaxEventInterpret*. This class is so named, because it is intended to represent a distinct interpretation of an event configuration (e.g. connecting particles to the decay tree according to one out of a number of hypotheses, applying different jet energy corrections, etc.). To facilitate the development of numerous parallel or subsequent event interpretations, the *PaxEventInterpret* class features a copy constructor, which provides a deep copy of the event container with all data members, physics objects, and their redirected relations.

The PAX toolkit offers a file I/O scheme for persistent storage of the event container, based on STL streams. It allows the user to write the contents of *PaxEventInterpret* instances with all contained physics objects as well as their relations to PAX data files. When restoring the data from file, an empty *PaxEventInterpret* instance is filled with the stored data and objects and all object relations are reproduced.

The PAX data file format is multi-version and multi-platform compatible and consists of binary data chunks, that allow file structure checks and fast positioning.

PAX also provides the possibility to write/read *PaxEventInterpret* instances to/from strings. This way, the user can store PAX objects to any data format supporting strings or binary data fields like databases or experiment specific data formats.

All classes of the PAX kernel can be used in compiled mode within the C-interpreter of ROOT. A more detailed description of the PAX kernel functionalities can be found in reference [4].

THE PAX ACCESSORIES

The PAX factory

The PaxFactory is an accessory to the PAX kernel that facilitates the bookkeeping of event hypothesis evolution.

The class *PaxProcess* is designed to allow the evolution of different combinatorial hypotheses (*event interpretations*) of an event according to a certain physics process. This task arises during the a priori ambiguous partonic reconstruction of processes with multiple reconstructed objects of the same type. Figure 1 gives single top production with leptonic top decay for illustration: permuting the plotted three jets from figure 1.b at the t→Wb vertex in figure 1.a provides three different hypotheses. In addition, the normally two-fold ambiguity of the longitudinal neutrino momentum, as provided by a W-mass constraint, doubles the number of combinatorial hypothesis for the t-quark. With the *PaxProcess* class, the analyst can store and manage an arbitrary number of event-interpretations including their physics objects and relations. Like all other PAX objects, a *PaxProcess* instance allows to store data in the user records and can record pointers to an arbitrary number of instances of arbitrary C++ classes.

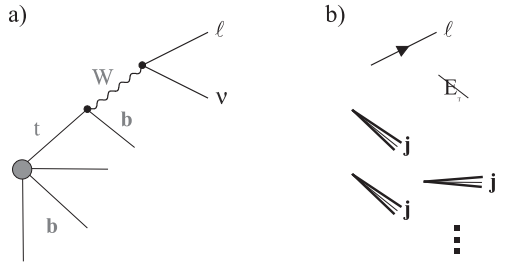


Figure 1: a) Schematic view of single top production with leptonic top decay. b) The visible reconstructed partons of this channel.

A higher degree of automation at the evolution of combinatorial hypotheses is provided by the class *PaxAutoProcess*. This derivative of the *PaxProcess* class features automatic evolution of all possible combinatorial hypotheses of an event. The rules, according to which these hypotheses are evolved, are defined by user-defined static process-model, that is passed to the *PaxAutoProcess* instance at construction time. This process-model is a *PaxEventInterpret* instance containing a prototype of the process decay-chain with parameters customizing the behaviour of the *PaxAutoProcess* class. The remaining step, to be done by the analyst, is to further process the evolved event interpretations in standard decision techniques, for instance.

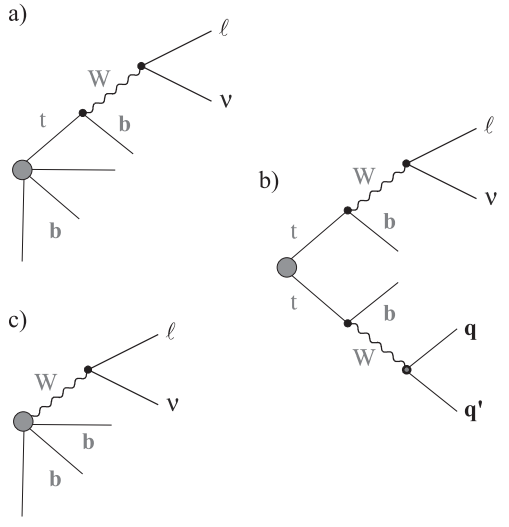


Figure 2: Schematic view of a) single top production and its main backgrounds b) top-pair production and c) jet-associated W production.

A further aspect of hypothesis evolution, besides the resolution of combinatorial ambiguities, is the parallel evolution of different physics process hypotheses of an event. As illustrated in figure 2 for the analysis of single top pro-

duction, the analyst might want to distinguish the signal channel (figure 2.a) from its main backgrounds individually (figure 2.b and 2.c).

To allow easy management of different physics process hypotheses of an event, the The class *PaxProcessFactory* provides storage and easy access to an arbitrary number of processes (i.e. instances of the class *PaxProcess* or derivatives) as well as user records and recording of arbitrary C++ pointers.

While the *PaxEventInterpret* instances with their physics objects are intented to remain in memory for one event, the classes of the PaxFactory accessory are designed for lifetimes of up to one computing job. Therefore, the classes *PaxProcess* and *PaxProcessFactory* provide virtual member functions to be called at the beginning and end of a job, at the beginning and end of a run, and, of course, at event analysis and event finishing time. By default, the methods of the *PaxProcessFactory* class invoke the corresponding methods of all managed *PaxProcess* instances.

The PaxFactory accessory provides the class *PaxEventInterpretTTree*, which allows to automatically copy selected observables from *PaxEventInterpret* instances on an event-by-event basis into the TTree of a ROOT file. Those observables may be kinematic data or user records of certain contained physics objects as well as user-records of the event-interpretation. The automatic copy is performed according to a user-defined, static copy-model in the form of a *PaxEventInterpret* instance, that is passed to the *PaxEventInterpretTTree* instance at construction time.

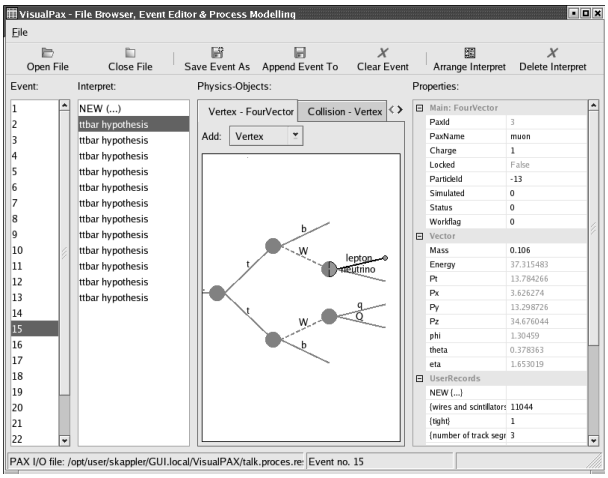


Figure 3: The Graphical User-Interface of VisualPax allows browsing of PAX I/O files and editing of *PaxEventInterpret* instances.

Visual PAX

VisualPax is a recenlty developed accessory to the PAX kernel that allows browsing of PAX I/O files and editing of *PaxEventInterpret* instances in a Graphical User-Interface. VisualPax is based on the wxWidgets open source, cross-platform native user-interface framework [8]. As shown

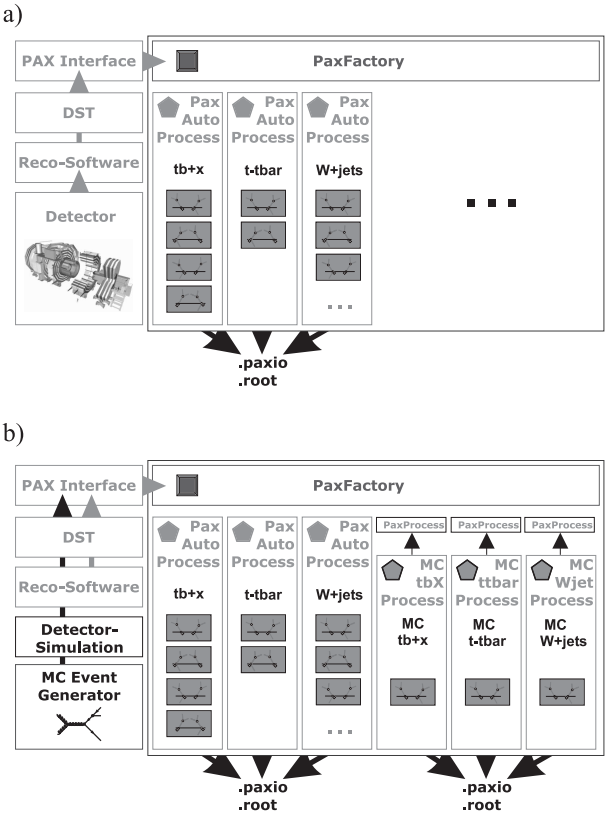


Figure 4: Example scheme for the use of the PAX kernel plus accessories in advanced physics analyses. The same software is used in different configurations when running over a) experiment data and b) Monte-Carlo data.

in figure 3, VisualPax allows to graphically display and modify event interpretations including properties and decay chains of the contained physics objects. Therefore, with the help of VisualPax and PAX I/O files, the process-models for *PaxAutoProcess* instances as well as the copy-models for *PaxEventInterpretTTree* instances can be managed in a comfortable way.

PAX ADVANCED ANALYSIS EXAMPLE

Advanced physics analyses realized with the PAX kernel and accessories can be designed according to the schema shown in figure 4.

When running over experiment data, a dedicated, experiment-specific interface class for filling the PAX containers (i.e. *PaxEventInterpret* instances) represents the interface between detector reconstruction software and the PAX factory. Once all relevant information is filled, the objects are passed to a *PaxProcessFactory* instance that manages *PaxAutoProcess* instances for each of the physics processes under study. Each of the *PaxAutoProcess* instances now evolves the combinatorial hypotheses for each event according to its process-model, that has been prepared earlier, e.g. with VisualPax. The virtual method *finishEvent()* of the *PaxProcessFactory* class then can be used to process

the resulting event hypotheses of all processes with decision techniques such as Likelihood methods, Neural Networks, Decision Trees etc. The results of the analysis can be written to PAX I/O files or selected observables can be written to a TTree of a ROOT file by using the *PaxEventInterpretTTree* class.

When running over Monte-Carlo data, the generator information can be passed to the PAX factory in addition. Furthermore, the *PaxProcessFactory* can be extended by *PaxProcess* derivatives exploiting the Monte-Carlo truth in order to train the deployed decision techniques in terms of ambiguity resolution and background-process suppression.

VisualPax can be used at any stage of the analysis to redefine process-models or monitor the results.

PAX PROJECT INFRASTRUCTURE

The PAX kernel and its officially supported accessories are continuously maintained and further developed by currently eight core developers and undergoes regular quality ensurance [9]. The PAX webpage [10] provides the PAX Users Guide [11], a comprehensive text documentation of the PAX toolkit, as well as class reference and fast navigator pages for download or online use. Version management of the software project is handled with a web-browsable Version Control System (CVS) [12][13].

ACKNOWLEDGEMENTS

The authors would like to thank Dominic Hirschbuehl, Yves Kemp, Patrick Schemitz, and Thorsten Walter, for helpful contributions and feedback.

REFERENCES

[1] M. Erdmann et al., *Physics Analysis Expert*, Proceedings of the 14th Topical Conference on Hadron Collider Physics, HCP2002, Karlsruhe, Germany, 2002.

[2] M. Erdmann, D. Hirschbühl, C. Jung, S. Kappler, Y. Kemp, M. Kirsch et al., *Physics Analysis Expert PAX: First Applications*, Proceedings of the conference on Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, USA, physics/0306085, 2003.

[3] M. Erdmann, U. Felzmann, D. Hirschbühl, C. Jung, S. Kappler, M. Kirsch et al., *New Applications of PAX in Physics Analyses at Hadron Colliders*, Proceedings of the conference on Computing in High Energy and Nuclear Physics (CHEP04), Interlaken, Switzerland, 2004.

[4] S. Kappler, M. Erdmann, U. Felzmann, D. Hirschbühl, M. Kirsch, G. Quast et al., *The PAX Toolkit and its Applications at Tevatron and LHC*, IEEE Trans. Nucl. Sci., vol. 53, no. 2, 2006.

[5] B. Stroustrup, *The C++ Programming Language*, Addison Wesley, ISBN 0-201-88954-2, 1997.

[6] *CLHEP library*, documentation online available: http://proj-clhep.web.cern.ch/proj-clhep/, 2006.

[7] R. Brun et al., *ROOT, an object oriented data analysis framework*, Proceedings of the 23rd CERN School of Computing, Marathon (Greece), 2000.

[8] *wxWidgets: open source, cross-platform native UI framework*, online available: http://www.wxwidgets.org/, 2004-2006.

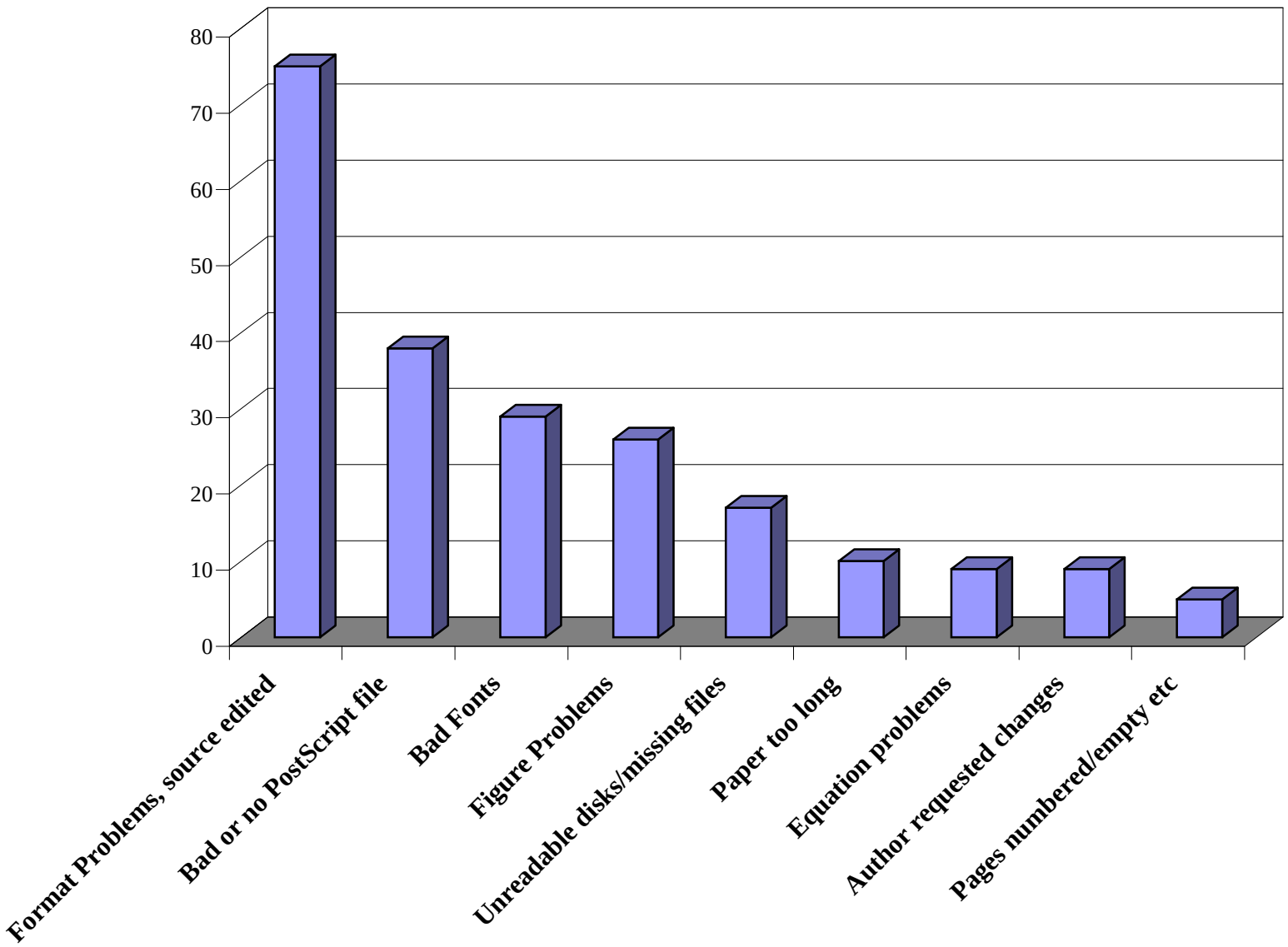
[9] *Borland Together Architect*, Borland Software Corporation, 20450 Stevens Creek Blvd., Cupertino, CA 95014, USA, 2005.

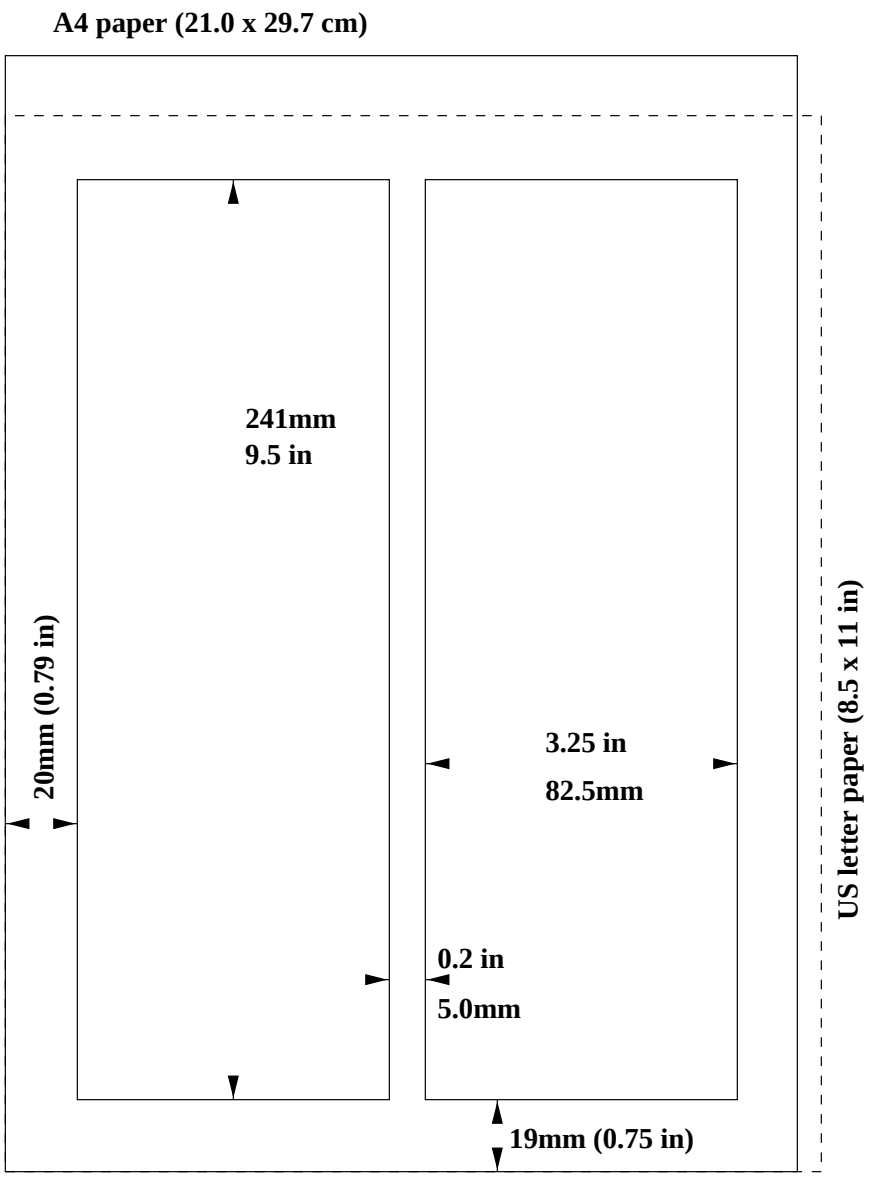
[10] M. Erdmann, S. Kappler, M. Kirsch, A. Schmidt, *PAX – Physics Analysis eXpert*, online documentation and support: http://cern.ch/pax, 2006.

[11] M. Erdmann, S. Kappler, M. Kirsch, A. Schmidt, *PAX Users Guide*, online available: http://cern.ch/pax, 2006.

[12] *CVS - Concurrent Version System*, documentation online available: http://www.nongnu.org/cvs/, 2006.

[13] *PAX CVS Repository*, online available: http://isscvs.cern.ch/cgi-bin/viewcvs-all.cgi/?cvsroot=pax, 2006.


80
70
60
50
40
30
20
10
0
Format Problems, source edited
Bad or no PostScript file
Bad Fonts
Figure Problems
Unreadable disks/missing files
Paper too long
Equation problems
Author requested changes
Pages numbered/empty etc


A4 paper (21.0 x 29.7 cm)
241mm
9.5 in
20mm (0.79 in)
3.25 in
82.5mm
0.2 in
5.0mm
19mm (0.75 in)
US letter paper (8.5 x 11 in)