



Implementation of new Features into the Atom Framework

Dominik Neuenfeld, University of Heidelberg, Germany

September 8, 2012

The reuse of HEP analyses performed at large experiments like ATLAS or CMS might help to exclude new particle physics models, even if the analyses are not tailored for them. This allows for the exclusion of models at an early stage and to accelerate the development of phenomenologically interesting ones. `emphAtom` is the name of a tool currently under development which aims to provide a framework to apply HEP analyses to new models on particle level in order to find exclusion limits. In the course of the DESY summer school various features were implemented into Atom. This report serves as documentation of the enhancements and changes that have been made to the code.

Contents

1	Introduction	3
1.1	Description of Atom	3
2	Changes to the Atom Code	4
2.1	Data Processing	5
2.2	Output	7
2.2.1	Python Plugin System	7
2.2.2	Visualisation of Output	8
2.3	Statistics	9
2.3.1	Python Plugin System	12
3	Outlook and Conclusion	14
3.1	Outlook	14
3.2	Conclusions	15
4	Acknowledgements	15
5	References	16

1 Introduction

Since the LHC has started operation in 2009, no convincing hints for Supersymmetry or other BSM physics were found. Instead, the results indicate that the Standard Model of Particle Physics is the correct description up to energies that can be reached by the LHC.

However, there is strong evidence that the Standard Model can not be the final answer in the search for a theory of everything. Consequently, many models have been, and still are being developed to overcome the problems of the Standard Model while containing it as a low energy effective theory.

At the LHC experiments, much effort is put into constraining the parameter space of new models like Supersymmetry or additions to the Standard Model like Axions. However, usually the analyses are only realised for a specific model and thus it is hard to draw conclusions for other models. Meanwhile, new models are developed which pend to be tested against real data.

It is obvious that untested models might be sensitive to analyses that already have been realised, and hence reusing old data might offer the possibility to restrict the parameter space without having to implement a fully-fledged analysis.

This report discusses the work on Atom, a framework that is designed to check new models against existing analyses. I have decided to turn this report into a small documentation on the changes that have been done to the Atom code, and I hope that the reader excuses (or acknowledges) the fact that this report is rather technical.

1.1 Description of Atom

Atom [1] (“A Test Of Models”, name preliminary) is a framework build upon Rivet[2] which allows for an estimate answer to the question whether a physics model will probably be excluded by experiments. As the base version of Rivet, Atom works on the particle-level taking events in the .hepmc format [3] as the definition of a model. Although this is a disadvantage in terms of precision, it allows for a fast execution of analyses and still leads to an acceptable accuracy. Figure 1.1 shows a comparison between ATLAS, PGS and Atom data.

Rivet, which is usually used for monte carlo validation, provides methods for a fast processing of analyses. This allows for a uncommon ansatz for testing models. Different to the “classical” approach of selecting some analyses which are sensitive to the model to be tested and implementing them, Atom can run all available analyses. Certainly, there

2 Changes to the Atom Code

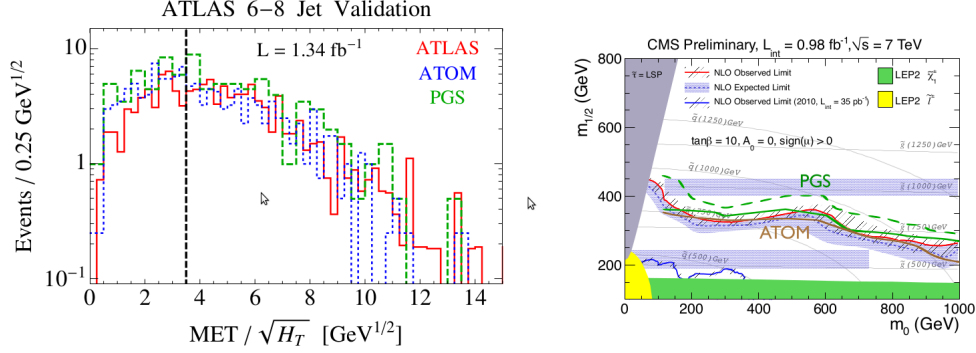


Figure 1: The left plot features a validation of a kinematic plot for Atom. It shows the missing energy significance in the ATLAS 6-8 jets plus missing energy search, for the MSUGRA benchmark point with $m_0 = 1220$ GeV and $m_{1/2} = 180$ GeV, $\tan\beta = 10$, and $A_0 = 0$ GeV. The right plot is an exclusion limit validation plot for Atom. It shows the CMS limit for the Same-Sign dilepton search by CMS, and superimposed the PGS (green) and Atom (brown) curves. (taken from [1])

will be analyses which do not give good results due to signal leakage in the background region. When working with Atom this has to be kept into mind.

Atom provides methods to track the signal efficiencies at any stage of the analysis for single subprocesses and the total events. It also calculates the logarithmic derivative of the signal efficiency with respect to the cut position. This information can be used to warn the user if one encounters large uncertainties in the final efficiencies, e.g. if cuts are applied on steeply falling signal distributions.

Furthermore, Atom extends the set of Rivet final states and provides many definitions of objects used in CMS and ATLAS analyses like isolated particles. This makes it especially easy to implement CMS and ATLAS analyses in Atom.

2 Changes to the Atom Code

During the course of this project, different changes were made to the the Atom code which I will outline in this section. In the process of development the following design principles were used for guidance (in arbitrary order)

- Readability
In case it was necesseay to make a compromise between performance and clarity

of the code, clarity was preferred. This is partly owed to the fact that the code lacks a proper documentation, so reading the code should be as easy as possible. Further, clearly structured code is easier to debug.

- **Independence from Rivet**
A strong entanglement between Atom and Rivet makes Atom very sensitive to changes in Rivet, which is developed independently. Hence, changes to the Rivet code may cause issues in Atom. To prevent this, it has been avoided to make changes to the Rivet code.
- **Few Dependences**
When it comes to releasing Atom, it is important that the installation is as easy as possible. One aspect of making the installation easier is to reduce the dependencies of Atom on third person libraries and packages. Therefore no new libraries were used but the ones already used by Rivet.
- **Modularity**
In order to allow for changes inside Atom without having to rewrite large parts of the code, interfaces were heavily used. This reduces the mutual dependence of different units of the Atom code.

2.1 Data Processing

The data processed and generated by Atom can be roughly grouped into the following four categories.

1. Event Data
2. Process Data
3. Analysis Data
4. Statistical Data

This distinction is implemented in Atom using the classes `Atom::EventMetadata`, `Atom::ProcessMetadata`, `Atom::AnalysisMetadata` and `Atom::StatisticalMetadata`, respectively, all located in `/include/Atom/EventMetadata.hh`.

Atom::EventMetadata contains everything associated with an event specified in the input *.hepmc* file. For storing multiple instances of EventMetadata a new type is defined

```
typedef <int, Atom::EventMetadata> AtomEvents
```

The key of this map is set to the event id.

The class Atom::ProcessMetadata stores information about hard processes, which is part of an hepmc event. These are in particular subprocesses and the corresponding particle ids. The difference between the event and process data is that the process is only defined by its Feynman diagram, whereas the event contains information on kinematics, decay chains, hadronisation, etc. Furthermore, the partial cross sections are stored for each process. For handling of multiple processes

```
typedef <int, Atom::ProcessMetadata> AtomProcesses
```

is defined, where the process id is used as the map key, uniquely identifying the hard process. Conventionally, the process id “0” is used to store information of all processes.

Atom::AnalysisMetadata contains data which is generated by the used analyses. This includes the efficiencies and cuts (also including cut efficiencies), information on the final states, and information on whether an event passed the analysis or not. For easier processing, methods were added which allow access to the metadata stored in the *.info* file of the analysis.

```
typedef <std::string, Atom::AnalysisMetadata> AtomMetadata
```

The key is equal to the unique name of the analysis.

Statistical data is intended to include everything generated by statistic modules. It is stored in Atom::StatisticsMetadata. Subcontainers for the number of signal events, CL_b , CL_{s+b} and CL_s are already implemented.

```
typedef <std::string, Atom::StatisticsMetadata> AtomStatistics
```

Again, the key is name name of the analysis.

Each of these container classes has provides a bitflag to check which information it contains. This is design to be used by later versions which also include a method to load data from a file (c.f. section 3.1).

Atom now provides the interface Atom::AtomData which gives access to the data collected/generated by Atom. This interface was introduced to separate the internal structure

of Atom from the Python plugin systems. `Atom::AtomData` provides methods to access the container classes specified above, and hence the Python plugins only have to know the interface instead of any specific class internal to Atom.

None of the classes supports buffered processing and hence the way the information is stored relies heavily on the assumption that the generated data does not get too large. It has to be checked whether this assumption is appropriate.

2.2 Output

Rivet itself saves rarely any data except histograms, generated by the analysis. In the current version of Rivet, Aida[?] is used to save the histogrammed data in an XML-like format. As Aida is not particularly well suited to save large amounts of information or non-histogram information and secondly will be removed from Rivet version 2.0 in favour of YODA[?], the Root library[4] had been used to store the information. While the Root-libraries are well suited for saving large amounts of information, using them obviously introduces a Root dependence into Atom.

This, however, is a disadvantage for two reasons. First, the user does not only have to install Rivet but also Root, and second this makes the Atom install unnecessary large and unappealing to people who do not use Root.

Therefore, a plugin based output system has been developed. Atom only provides an interface to Python, which allows to access the data provided by Atom. Having the possibility to write classes handling the output in Python is very powerful, as Python provides many modules which make it easy to process the data. Furthermore, it allows to add new functionality to Atom without having to recompile the code. This is especially useful for users which need a specific output format and/or do not need all data generated by Atom.

In order to have a standardised output if necessary, Atom provides tags which can be accessed in C++ (`#include "/include/Atom/AtomTags.hh"`) and Python (`from atom_tags import *`).

2.2.1 Python Plugin System

For the implementation of the interface the SWIG [5] framework was chosen. The benefit is that at least in principle SWIG is not needed to build Atom.¹ Furthermore, SWIG does not only allow for an interface to Python, but many more programming and scripting languages.

¹It in fact is necessary as the Rivet built depends on it, although this dependence could be removed.

All output modules (whether C++ or Python) have to inherit from `Atom::AnalysisWriter` (`atom.analysisWriter`, respectively), which provides the abstract method `void writeData (AtomData* data)`. This method has to be overridden in any derived class. It is called after all analyses have been processed and should contain code which displays data from the `Atom::AtomData` interface.

Listing 1 shows how a simple writer can be implemented

Listing 1: Python

```
1 from atom import AnalysisWriter
2
3 def crop(s):
4     return s[:s.find("\x00")]
5
6 class ExampleWriter(AnalysisWriter):
7     # override this function to display the data
8     def writeData(self, data):
9         print "-----\n"
10        print " Example Writer output \n"
11        print "-----\n\n"
12        for k,v in data.getAnalysisMetadata().iteritems():
13            print "Analysis " + str(k) + "\n"
14            for cut in v.getCutInfo().iteritems():
15                print str(crop(cut.name)) + " has efficiency " + str(cut.cuteff) +
                    "\n"
```

The loading and execution of selected writers is done by a C++ implementation of `Atom::AnalysisWriter` called `Atom::PythonPluginWriter`. It handles the initialisation of Python and the Python calls from C++. Recently, the class `Atom::PythonWrapper` has been added (together with the statistics plugin system), which provides easier methods for handling Python. `Atom::PythonPluginWriter` should inherit from `Atom::PythonWrapper` to have the dependence of Atom on Python in the class `Atom::PythonWrapper`.

A small caveat about the use of Python in Atom should be mentioned. As the executable file which starts Atom is a Python script, Python is already initialised when the C++ code is called. In order to leave the Rivet code as it is, the writers are called via `std::atexit`. However, if `std::atexit` is invoked, the Python runtime has already been shutdown and thus Python has to be started again.

2.2.2 Visualisation of Output

To make the output human-readable, XML [6, 7], XSLT [8, 9] and JavaScript[] were used. The benefit of the chosen method is that it needs nothing but a modern webbrowser to

view the results. Anticipating the possible implementation of reading stored data back to Atom (c.f. 3.1) the XML filetype is well suited. It allows to display the data via XSLT while being easy to parse, especially since a SAX parser is shipped with Python (import xml.sax). Besides, a XML/XSLT/XHTML output allows navigation and linking of plots in the analysis report.

Alternatively, one could think about a pure HTML output, which has the benefit of being just one file, while having the drawback that the stored information is harder to read from the file. Also a .pdf or LaTeX output is possible. However, they also suffer the problem that it is more difficult to write a reader.

In a tentative implementation which can be found in `python/readableOutput.py` the definitions from the `atom_tag` module are used for the XML tags. From a point of view of re-readability this is a good idea. However, the current `atom_tag` implementation features unnecessarily long tag names and hence should be replaced by a version with shorter tags. When writing a new `atom_tag` version (i.e. changing `/include/Atom/AtomTags.hh`) one should alter the tag `ATOM_TAG_VERSION`, which is intended to identify the used set of tags. It might be a good idea to change the current implementation which uses `#define` directives to static class members, to allow for support of many tag versions.

The display of the data is taken over by an XSLT script defined in the file `/Tools/xmldisplay.xsl` which at the moment has to be copied manually to the folder containing the generated .xml file. It is linked directly in the XML file

An example of the generated output is shown in figure 2.

The XSLT script contains Javascript to handle collapsing of cut, efficiencies, and the detail section of all the applied analyses. An advantage of XSLT is that it is quite powerful and allows to arrange data according to information stored in XML tags.

For example the simple graphical realisation of the cutflow is realised exclusively in XSLT. The XML data is shown in listing 2.

As one can see, no tree structure is evident. Only the cut tags contain information about their parent nodes. This makes it easy to load the XML file in the containers provided by Atom. However, listing 3 shows how the XML file is turned into a tree-like structure using recursion in XSLT as can be seen in figure 2.

The display of mathematical formulas is done with MathML [?].

2.3 Statistics

The statistic abilities of Atom are still under development, as it has not been decided yet, which functionality should eventually be provided by Atom. To maintain modularity

2 Changes to the Atom Code

Atom Analysis Report

version 0.1

1	ATLAS_2012_CONF_2012_033	2-6 jets + MET SUSY search at 7TeV	(click to collapse)																																																																																																				
<div><div>Information</div><div>(click to collapse)</div><div><div>Description</div><div><Insert a fairly long description, including what is measured and if possible what it is useful for in terms of MC validation and tuning. Use \LaTeX for maths like \sqrt{s} > \\unit{50}{GeV}. Use single quotes around the block if required (see YAML 1.2 manual)></div><div>References</div><div>Luminosity</div><div>4700.0⁺</div><div>Cross section</div><div>0.0868835241371</div><div>Corrected cross section</div><div>0.09890182662700299</div></div></div>																																																																																																							
<div><div>Cuts</div><div>(click to expand)</div></div>																																																																																																							
<div><div>Efficiencies</div><div>(click to expand)</div></div>																																																																																																							
2	CMS_PAS_SUS_11_015	<Insert short CMS_PAS_SUS_11_015 description>	(click to expand)																																																																																																				
3	CMS_PAS_SUS_12_011	<Insert short CMS_PAS_SUS_12_011 description>	(click to collapse)																																																																																																				
<div><div>Information</div><div>(click to collapse)</div><div><div>Description</div><div><Insert a fairly long description, including what is measured and if possible what it is useful for in terms of MC validation and tuning. Use \LaTeX for maths like \sqrt{s} > \\unit{50}{GeV}. Use single quotes around the block if required (see YAML 1.2 manual)></div><div>References</div><div>Luminosity</div><div>4980.0⁺</div><div>Cross section</div><div>0.0868835241371</div><div>Corrected cross section</div><div>0.09890182662700299</div></div></div>																																																																																																							
<div><div>Cuts</div><div>(click to collapse)</div><table><tr><th>Cut Flow</th><th>Description</th><th>Efficiency</th><th>Derivative</th></tr><tr><td>Ht</td><td>description of Ht cut</td><td>20.00%</td><td>-0.767</td></tr><tr><td>....Mht</td><td></td><td>11.72%</td><td>-1.089</td></tr><tr><td>.....dphiJ1</td><td>description of dphiJ1 cut</td><td>11.53%</td><td>0.000</td></tr><tr><td>.....dphiJ2</td><td></td><td>10.54%</td><td>-0.164</td></tr><tr><td>.....dphiJ3</td><td></td><td>9.78%</td><td>0.000</td></tr><tr><td>.....lowHt</td><td></td><td>1.15%</td><td>0.382</td></tr><tr><td>.....lowHt1</td><td></td><td>0.61%</td><td>0.714</td></tr><tr><td>.....lowHt2</td><td></td><td>0.54%</td><td>0.000</td></tr><tr><td>.....lowHt3</td><td></td><td>0.00%</td><td>0.000</td></tr><tr><td>.....lowHt4</td><td></td><td>0.00%</td><td>0.000</td></tr><tr><td>.....medHt</td><td></td><td>1.73%</td><td>0.000</td></tr><tr><td>.....high1Ht</td><td></td><td>1.71%</td><td>0.000</td></tr><tr><td>.....high1Ht1</td><td></td><td>0.65%</td><td>0.000</td></tr><tr><td>.....high1Ht2</td><td></td><td>0.79%</td><td>1.894</td></tr><tr><td>.....high1Ht3</td><td></td><td>0.28%</td><td>-5.356</td></tr><tr><td>.....high2Ht</td><td></td><td>1.83%</td><td>0.240</td></tr><tr><td>.....high2Ht1</td><td></td><td>0.80%</td><td>0.000</td></tr><tr><td>.....high2Ht2</td><td></td><td>1.03%</td><td>-1.862</td></tr><tr><td>.....high3Ht</td><td></td><td>3.36%</td><td>-4.281</td></tr><tr><td>.....high3Ht3</td><td></td><td>3.36%</td><td>-1.299</td></tr><tr><td>.....medHt1</td><td></td><td>0.20%</td><td>2.246</td></tr><tr><td>.....medHt2</td><td></td><td>1.15%</td><td>0.383</td></tr><tr><td>.....medHt3</td><td></td><td>0.19%</td><td>0.000</td></tr><tr><td>.....medHt4</td><td></td><td>0.19%</td><td>0.000</td></tr></table></div>				Cut Flow	Description	Efficiency	Derivative	Ht	description of Ht cut	20.00%	-0.767Mht		11.72%	-1.089dphiJ1	description of dphiJ1 cut	11.53%	0.000dphiJ2		10.54%	-0.164dphiJ3		9.78%	0.000lowHt		1.15%	0.382lowHt1		0.61%	0.714lowHt2		0.54%	0.000lowHt3		0.00%	0.000lowHt4		0.00%	0.000medHt		1.73%	0.000high1Ht		1.71%	0.000high1Ht1		0.65%	0.000high1Ht2		0.79%	1.894high1Ht3		0.28%	-5.356high2Ht		1.83%	0.240high2Ht1		0.80%	0.000high2Ht2		1.03%	-1.862high3Ht		3.36%	-4.281high3Ht3		3.36%	-1.299medHt1		0.20%	2.246medHt2		1.15%	0.383medHt3		0.19%	0.000medHt4		0.19%	0.000
Cut Flow	Description	Efficiency	Derivative																																																																																																				
Ht	description of Ht cut	20.00%	-0.767																																																																																																				
....Mht		11.72%	-1.089																																																																																																				
.....dphiJ1	description of dphiJ1 cut	11.53%	0.000																																																																																																				
.....dphiJ2		10.54%	-0.164																																																																																																				
.....dphiJ3		9.78%	0.000																																																																																																				
.....lowHt		1.15%	0.382																																																																																																				
.....lowHt1		0.61%	0.714																																																																																																				
.....lowHt2		0.54%	0.000																																																																																																				
.....lowHt3		0.00%	0.000																																																																																																				
.....lowHt4		0.00%	0.000																																																																																																				
.....medHt		1.73%	0.000																																																																																																				
.....high1Ht		1.71%	0.000																																																																																																				
.....high1Ht1		0.65%	0.000																																																																																																				
.....high1Ht2		0.79%	1.894																																																																																																				
.....high1Ht3		0.28%	-5.356																																																																																																				
.....high2Ht		1.83%	0.240																																																																																																				
.....high2Ht1		0.80%	0.000																																																																																																				
.....high2Ht2		1.03%	-1.862																																																																																																				
.....high3Ht		3.36%	-4.281																																																																																																				
.....high3Ht3		3.36%	-1.299																																																																																																				
.....medHt1		0.20%	2.246																																																																																																				
.....medHt2		1.15%	0.383																																																																																																				
.....medHt3		0.19%	0.000																																																																																																				
.....medHt4		0.19%	0.000																																																																																																				

2 Changes to the Atom Code

Efficiencies		(click to collapse)
Description	Efficiency	Signal Events
Efficiency of the 1-1 search	0.61% ^{+0.23%} _{-0.23%}	3
Efficiency of the 1-2 search	0.54% ^{+0.39%} _{-0.39%}	3
Efficiency of the 1-3 search	0.00% ^{+0.00%} _{-0.00%}	0
Efficiency of the 1-4 search	0.00% ^{+0.00%} _{-0.00%}	0
Efficiency of the 2-1 search	0.20% ^{+0.14%} _{-0.14%}	1
Efficiency of the 2-2 search	1.15% ^{+0.54%} _{-0.54%}	6
Efficiency of the 2-3 search	0.19% ^{+0.19%} _{-0.19%}	1
Efficiency of the 2-4 search	0.19% ^{+0.19%} _{-0.19%}	1
Efficiency of the 3-1 search	0.65% ^{+0.41%} _{-0.41%}	3
Efficiency of the 3-2 search	0.79% ^{+0.36%} _{-0.36%}	4
Efficiency of the 3-3 search	0.28% ^{+0.21%} _{-0.21%}	1
Efficiency of the 4-1 search	0.80% ^{+0.42%} _{-0.42%}	4
Efficiency of the 4-2 search	1.03% ^{+0.49%} _{-0.49%}	5
Efficiency of the 5-1 search	3.36% ^{+0.81%} _{-0.81%}	17
Efficiency of the Baseline Selection	9.78% ^{+1.33%} _{-1.33%}	48

Figure 2: Example of an preliminary version of an Atom report. As can be seen in the image the analyses and the details can be expanded and collapsed seperately. The used analyses we selected randomly and applied to a .hepmc with unknown content. The cutflow is presented in a primitive way and might be changed in the future. Please keep in mind that this output is not final yet. Please also appreciate the modern and minimalist design.

Listing 2: excerpt of readable.xml

```

1 <Cuts>
2   <Cut>
3     <name>Ht</name>
4     <idx>0</idx>
5     <parent_idx>0</parent_idx>
6     ...
7   </Cut>
8   <Cut>
9     <name>MHt</name>
10    <idx>1</idx>
11    <parent_idx>0</parent_idx>
12    ...
13  </Cut>
14  <Cut>
15    <name>dphiJ1</name>
16    <idx>2</idx>
17    <parent_idx>1</parent_idx>
18    ...
19  </Cut>
20  ...
21 </Cuts>

```

and allow the user to define his or her own analyses which can be run by Atom, a plugin system for statistic tools is necessary, independent of the functionality Atom might offer in the future.

2.3.1 Python Plugin System

The statistics plugin system of Atom is basically implemented analogous to the writer plugin system (c.f. section 2.2). The only difference lies in that the separation between code depending on and independent of Python is even stronger. However, the design should be carried over to the writer plugin system.

The interface `Atom::IStatisticsPlugin` (include/Atom/Interfaces/IStatisticsPlugin.hh), which is realized as an abstract class, is used as a base class for all plugins no matter if they are written in C++ or Python. The interface provides the method `doStatistics (Atom::AtomData*, Atom::IStatisticsWriter*)`, which gives an `Atom::AtomData` object to access the data generated by Atom and a pointer to an object of type `Atom::IStatisticsWriter`.

Objects inheriting from `IStatisticsWriter` have to implement the methods and hence are responsible for storing or writing the data. In the current Atom implementation, this is

Listing 3: xmldisplay.xsl

```
1 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="
  1.0">
2
3 ...
4
5 <!-- idx=0 and parent_idx=0 indicate root node -->
6 <xsl:apply-templates select="*[idx='0' and parent_idx='0']" />
7
8 ...
9
10 <!-- apply template to all child nodes of current node -->
11 <xsl:apply-templates select="../*[parent_idx=current()/idx and idx > 0]"
    />
12 ...
13
14 <xsl:template match="Cut/name">
15 <xsl:param name="show"/>
16 <!-- for every parent up to the root node-->
17 <xsl:if test="../idx > 0">
18   <xsl:apply-templates select="../../Cut[idx=current()/../parent_idx]/
      name" >
19     <xsl:with-param name="show">0</xsl:with-param>
20   </xsl:apply-templates>
21   <!-- write .... -->
22   <xsl:text>....</xsl:text>
23 </xsl:if>
24 <!-- only write down the name of the currently selected node -->
25 <xsl:if test="$show > 0">
26   <xsl:value-of select="." />
27 </xsl:if>
28 </xsl:template>
```

Listing 4: Python

```
1 from atom import IStatisticsPlugin
2
3 class SignalPlugin(IStatisticsPlugin):
4     def doStatistics(self, data, writer):
5         for k,v in data.getAnalysisMetadata().iteritems():
6
7             writer.setSignalEvents();
```

done by the `Atom::WriterHelper` singleton.

3 Outlook and Conclusion

3.1 Outlook

Although Atom in principle is fully operable, there are still many things that should be done before releasing a version to the public.

Atom needs the possibility to store and provide access to experimental data like the background and signal events. This is very crucial in order to allow for statistical analyses.

Atom still needs a command line. By construction analogous to Rivet one might get around compiling Rivet before compiling Atom and thus might reduce the dependencies on third party programs such as SWIG. The commandline is further needed to allow the user to load plugins for writing data or doing statistics. The selection of data to be stored works via environment variables and also should be moved into a command line tool.

Simultaneously, one might think about getting away from the histogram handling of Rivet and implement a histogram output (self developed or third party library) analogous to `Atom::EfficiencyHelper` or `Atom::CutHelper`. Besides the fact that AIDA will be removed from Rivet anyway, histogram handling done by Atom increases flexibility and decreases the dependence on Rivet.

Another improvement would be to refactor the output plugin system according to the statistics plugin system. The use of interfaces makes the code more readable and relaxes dependencies between the writer part of Atom and the actual handling of the analyses.

Furthermore a reader plugin system analogous to the writer plugin system which provides methods to load data is useful. This serves two purposes. First, it allows to convert

between different output formats (which indeed is helpful as the dependence of Atom on ROOT is only optional) and second, it allows to do statistics independent from the analyses.

To be able to maintain and use Atom, the code should be moved from its current version control system SVN to another one which allows for fast and more user-friendly² version control system. A bugtracker needs to be set up to give the possibility to report and track bugs, esp. if bugs might affect analysis which have been done before. Last but not least, there is lots of room for improvements when it comes to the documentation.

3.2 Conclusions

Although Atom still is in development, it has been shown that it indeed can be used to set limits on physics models[1]. In the course of this project Atom has been extended, primarily in terms of usability. Nonetheless, further efforts are necessary before Atom can be made available to the public.

4 Acknowledgements

I would like to express my gratitude towards my supervisor Andreas Weiler who gave me the chance to participate in a recent and interesting Project. My thank also goes to Michele Papucci for his support and discussions about Atom.

²e.g. proper branching support

5 References

- [1] Michele Papucci, Joshua T. Ruderman, and Andreas Weiler. Natural SUSY Endures. 2011.
- [2] Andy Buckley, Jonathan Butterworth, Leif Lonnblad, Hendrik Hoeth, James Monk, et al. Rivet user manual. 2010.
- [3] Matt Dobbs and Jorgen Beck Hansen. The HepMC C++ Monte Carlo event record for High Energy Physics. *Comput.Phys.Commun.*, 134:41–46, 2001.
- [4] ROOT. A data analysis framework, 2012.
- [5] SWIG. Simplified wrapper and interface generator, 2012.
- [6] W3C. Extensible markup language (xml), 2012.
- [7] W3Schools. Xml tutorial, 2012.
- [8] W3C. Xsl transformations (xslt), 2012.
- [9] W3Schools. Xslt tutorial, 2012.