Documentation for C++ LaTeX assistant ZetoTex

Carl Schaffer

April 28, 2015

1 Introduction

ZetoTex provides an assistant to simplify getting analysis results into finished LaTeX documents. It provides a custom class designed to convert data objects inherent to **C++** into LaTeX code. This is done by a LaTeX Assistant which receives and processes the data contents and produces a ".tex" file providing custom commands usable for writing. This document will show how to use the Latex assistant and provide examples of output.

2 Getting and Using the Assistant

The ZetoTex-Package can be found on github:

`https://github.com/DeltiKron/ZetoTex`
use `git clone` to obtain a working copy of the master branch.
In order to use the assistant, the custom header `zetotex.cpp` needs to be included in your **C++**-project. this can be done using

```
#include "path/to/zetotex.cpp"
```

in your code header. This provides the `tex_assistant` class which will be used to handle all actions described in the following sections.
If you are not compiling your code, you will have to work out how to make your interpreter handle things correctly. The `ROOT` frameworks' `CINT` interpreter can make the classes and methods available using:

```
  gSystem->CompileMacro("/path/to/zetotex.cpp");
```

2.1 The `tex_assistant` Class

The interface consists of an object of the type `tex_assistant`. The constructor takes one argument, which is a string containing the path where the latex document containing the new commands should be created. An empty version of this file is recreated every time the `tex_assistant` constructor is called. The actual writing is done by calling methods inherent to the `tex_assistant` class. Example Constructor call:

```
tex_assistant ta = tex_assistant("./output/tex_sample.tex");
```

2.2 Single Value Commands

A command to represent a single number can be generated using the `.ncmdValue()` method. It can be called as:

```
ta.ncmdValue("commandWithFloat",1.002);
```

for float values or as

```
ta.ncmdValue("commandWithInteger",1);
```

which produces the following output in the ".tex" file:

```
\newcommand{\commandWithFloat}{1.0002}

\newcommand{\commandWithInteger}{1}
```

Invoking these commands in a document yields:

Command	Output
`\commandWithFloat`	1.0002
`\commandWithInteger`	1

2.3 Writing arrays using ncmdArray

Data available in `std::vector` form can be written to using `.ncmdArray()`. This command requires three arguments:

1. the command name as a string
2. a `std::vector` containing the column headers
3. a `std::vector (std::vector )` containing single `std::vector` 's for each line

Here is some sample output from first calling

```
ta.ncmdArray("SampleArray",header,data);
```

which produces the following in the ".tex" file:

```
\newcommand{\SampleArray}{
        \begin{tabular}{|c|c|c|c|c|c|}
          \hline Column 0 & Column 1 & Column 2 & Column 3 & Column 4 & Column 5 \\
          \hline 0 & 1 & 2 & 3 &   &  \\
          \hline 0 & 1 & 2 & 3 &   &  \\
          \hline 0 & 1 & 2 & 3 &   &  \\
          \hline 0 & 1 & 2 & 3 &   &  \\
          \hline
        \end{tabular}
}
```

and then calling `\SampleArray` within a table environment in the final LaTeX document.

Column 0	Column 1	Column 2	Column 3	Column 4	Column 5
0	1	2	3		
0	1	2	3		
0	1	2	3		
0	1	2	3		

Table 1: Sample array

Note that in this example, the `std::vector` containing the column header strings is larger than the data. In this case, the assistant automatically fills empty cells to the right of the existing data after printing a warning on the console.