# Using database reader plug-ins for VisIt

Max Knötig

September 28, 2010

**Abstract**

The author presents his work done from beginning of April until the end of September 2010 at *MPA* in the Group of `Fritz Röpke`. This was to write a database reader plug-in for the visualization program *VisIt*, in order to read in the output format of *LEAFS*. The work later was extended to two other database formats, namely the *tracer* file format of *LEAFS* and the flexible *LHC* file format used by `Fabian Miczek` and `Philipp Edelmann`.

# Contents

# Chapter 1

# Introduction

This document is meant to help you to use my VisIt plug-ins for your work. There are three plug-ins so far, have a look at the Chapters 2.4, 2.5, 2.6. I want to first mention the websites and other sources of information which helped me out

- `https://wci.llnl.gov/codes/visit/`
- `http://www.visitusers.org/`
- `http://www.visitusers.org/forum/forum.pl`

which has manuals covering a the dataanalysis, the Python interface and how to get data into VisIt. The first link is the official VisIt website. The second link leads to the VisIt Wiki. The third link is the user forum. The next link is the VisIt 1.10 Source Code Documentation. There is so far (28.09.2010) no newer version.

- `http://www.visitusers.org/visit/1.10.0/doxygen/`

# Chapter 2

# The plug-ins

## 2.1 Getting the plug-in

First I will show you how to download your plug-in for your version of *VisIt*. The tarballs lie within the supernova repository and you will need access to it. Check out a copy of the repository (or just parts of it) like:

- svn co http://www.mpa-garching.mpg.de/svn/noether-group/sncode/trunk

You can find the plug-ins in the subfolder

- `sncode/code/visit/version1.12`
- `sncode/code/visit/version2.0`

The plug-ins are packed and end with `tar.gz`. To install version 1.12 plug-ins type:

- `visit_plugin -i <plugin-name>`

where `<plugin-name>` is something like 'lhc' or 'snI'. For VisIt vers. 2.1 plug-ins please refer to 2.2. Do not include the '.tar.gz' in the plug-in's name commands. The `visit_plugin` is a script from your version of *VisIt*. It compiles the code inside the plug-in.tar.gz into a directory on your hard disk. Make sure this script is from the *same*installation (architecture and version number.) as the *VisIt* you later make plots with. The directory for compiled plug-ins is

- `~/.visit/linux-x86_64/plugins/databases`

The architecture `(linux-x86_64)` of the computer may be different in your case.

## 2.2 It does not work!

If the `visit_plugin` tool is not available, or fails for some reason, you may need to use the manual approach and follow these steps. Untar the plug-in tarball:

- `tar -xzf <plugin-name>.tar.gz`

To install:

- `cd <plugin-name>`
- `xml2makefile -clobber <plugin-name>.xml`
- `make`

On *AIX* und the *xlc* compiler (e.g. the cluser *vip*) you need to manually add *-q64* to the *LDFLAGS* in the Makefile.

## 2.3 Uninstalling the plug-in

To UNinstall...

- `visit_plugin -u <plugin-name>`

Alternatively, you can UNinstall the plugin by explicitly removing *all* of the relevant .so's from the .visit directory in your home directory:

- `cd ~/.visit/<arch>/plugins/databases`
- `find . -name '*<plugin-name>*.so' -exec rm {} \; -print`

## 2.4 The snI plug-in

The snI plug-in produces a plot when opening the *.000 file of a time step. To install it please refer to the Section 2.1. It has no MT (multi timesteps) included, and is the only plug-in *that does not support the timebar*. The plug-in reads in the scalars and eventually the velocity vectors. Please refer to the excellent tutorial *VisIt Getting Started Manual* for making your plot pretty. I will give a brief introduction to the Python scripts for making movies in Section 3.3. Fig 2.1 shows a deflagation model of a suoernova. It is a contour plot of the levelset at $lvlset = 0$, which corresponds to the flamefront.

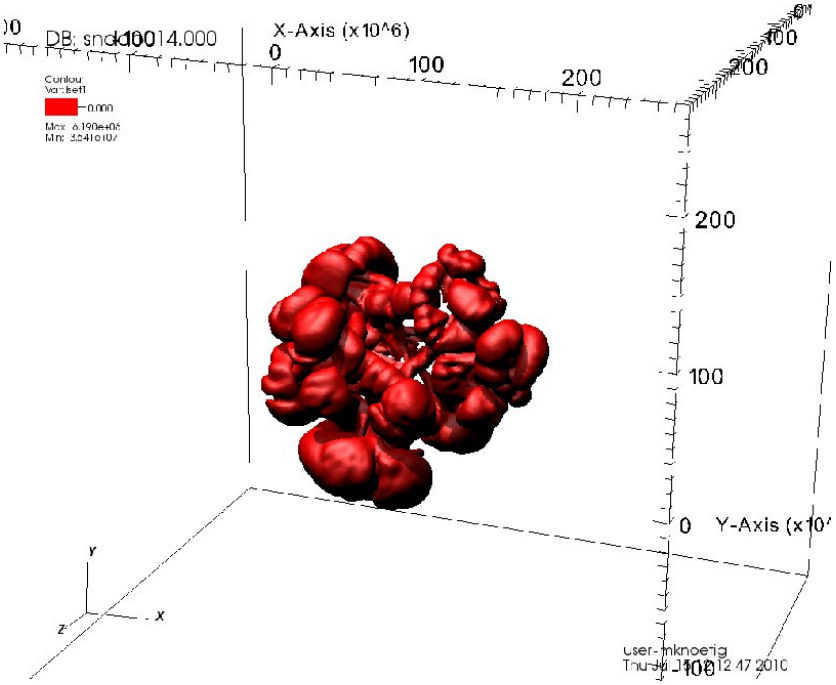


Figure 2.1: A supernova in VisIt. Contour plot of the levelset at $lvlset = 0$, which corresponds to the flamefront

## 2.5 The tracer plug-in

We use tracers in a post-process to calculate the abundances of ejecta in a supernova. These tracers are particles advected in the hydro code and form a irregular mesh. The tracer format is still under development may therefore be outdated when you read this. If so go to the Chapter 4 and change the code directly (but first ask around, whether somebody has an update). The plug-in reads the '.tracer' file and the time slider can be used. This plug-in is MT (multi time). You can start making your movies in the movie wizard, but it is also possible to script the process.
The new VisIt versions have brought some major improvements. The Selection window was added so users can create selections. A selection is a set of cells created by a plot. This feature is interesting for example to plot the cells over time which will become the most metal-abundant ones. Refer to the release notes for VisIt 2.1 for further information.
Fig. 2.2 shows a delayed detonation model. There is a transition from deflagation to detonation at about one second after ignition. The bubbles rising from the surface are expanding detonations.

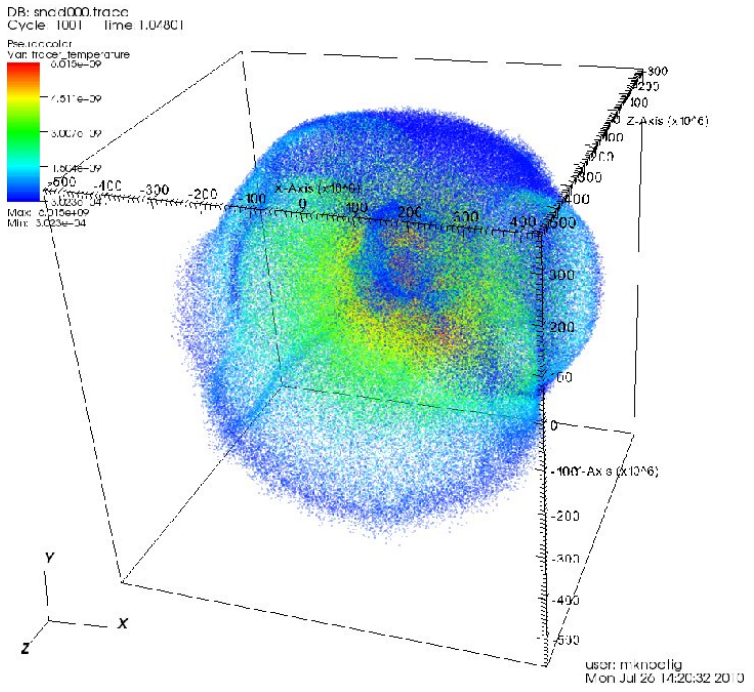


Figure 2.2: The tracers in VisIt. The plot shows the temperature of the tracers in a delayed detonation SNI model (DDT)

## 2.6 The lhc plug-in

The lhc format was introduced by Phillip and Fabian in order to specify a flexible data format. The plug-in was created using Philipp's C library for reading *.lhc files. The plug-in is single time(ST) but it is easy to activate the timebar. In order to scroll in time one needs to *group the files before opening them.* This is done in VisIt's *Select File* menu.

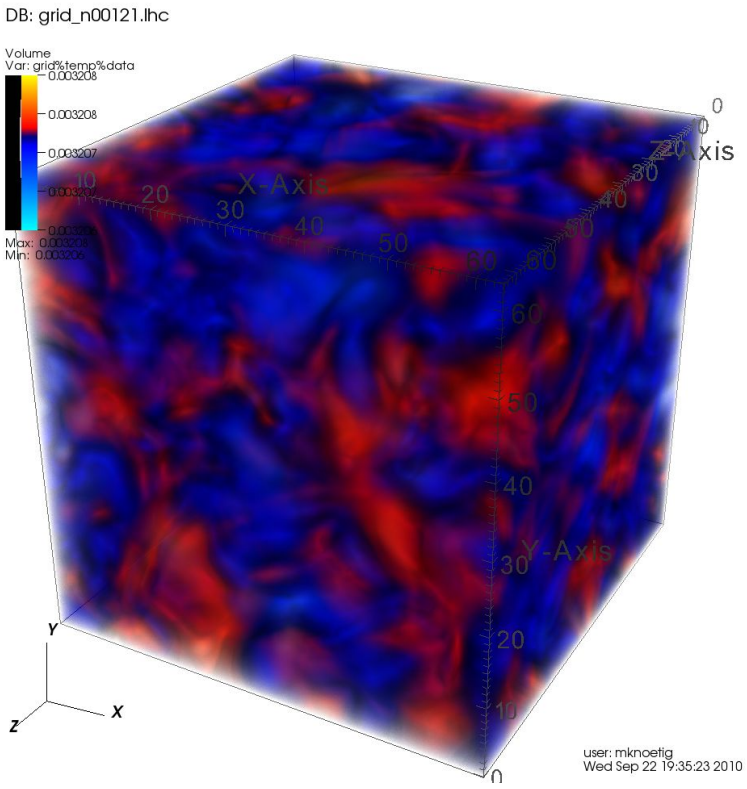


Figure 2.3: The lhc plug-in in action

VisIt is able to read VTK files (a standard 3D data format) and the *LHC* code has routines for producing those. This is a further option in getting data into VisIt: producing compatible files.

# Chapter 3

# Python interface

To speed up the process of plotting you may want to use Python with VisIt. This section explains how to start VisIt via Python and how to write and execute scripts. This Chapter covers a few parts of the VisIt *Python Interface* manual. The interface itself is subject to change, and future updates may add functionality to it.

## 3.1 Introduction

To start VisIt using the command line interface type:

- visit -cli

The functions in the GUI have Python bindings and you can use them in this interface. To load your data and plot eg. the density in a pseudocolor plot type:

```
OpenDatabase("<path to your data>/snddo015.000")
AddPlot("Pseudocolor","density")
DrawPlots()
```

This will open your data and add a plot to the stack of plots. The last command draws it. tTo alter the options of a plot or operator, add more windows, etc. . . look up the *Python Interface Manual*.
It is also possible to script the commands into one file and execute it. This will lead us to the movie scripts in the next sections. Type the following in order to execute a script

- `visit -cli -s <pathtoscript>/script.py`

## 3.2 Using a Python interpreter

Here are some notes about using the VisIt Python API with a standard Python interpreter (not the VisIt CLI that comes with VisIt). In order for Python to load VisIt's Python interface, it must know where to find visitmodule.so. There are different ways to do this but the simplest is to append to sys.path. The sys.path list contains the paths that Python will search when looking for

modules. To make sure that VisIt's Python module is located when we do 'import visit' or 'from visit import *', we need to append the path to VisIt's visit.so front end module. This can be done using the following code:

```
import sys
sys.path.append("<pathtovisit>/linux-x86_64/lib")
import visit
visit.Launch()
```

Note the use of `linux-x86_64` in the path. You may need to use linux-intel or darwin-i386, depending on your platform. There are two ways that you might import VisIt's symbols into Python. The first way adds the visit module to Python and requires you to add a visit. prefix to all function calls.

```
# you could do this:
import visit
visit.Launch()
```

The second way adds the visit module's symbols to the global Python namespace, mirroring what the VisIt CLI does.

```
# or, you could do this:
from visit import *
Launch()
```

One has to `Launch()` in the directory of the VisIt script.

## 3.3 Examples of scripts

In order to make a movie you need to produce a large amount of frames from your data. You could use the built in movie maker or use scripts. The following are two scripts I have written, which create a set of pictures along timesteps. This may be used in *ffmpeg*.

### 3.3.1 snI movie

There is a lot of existing Python code in the VisIt Wiki, have a look at

- `http://www.visitusers.org/index.php?title=Using_CLI`

This is how to make a movie out of your data. VisIt provides two ways to access a set of single time-state files as a single time varying database. The first method is a '.visit' file, which is a simple text file that contains the names of each file to be used as a time state in the time-varying database. The second method uses 'virtual databases', which allows VisIt to exploit the file naming conventions:

```
OpenDatabase("/usr/gapps/visit/data/wave.visit" 0)
OpenDatabase("/usr/gapps/visit/data/wave*.silo database")
```

This leads to the movie script for the *snI* output format. First, create a *.visit file containing all the timesteps that you need in order to make a smooth movie. This looks like:

```
 <path to snI>/snddo030.000
 <path to snI>/snddo031.000
 .
 .
 .
```

An example script script which will render the enery and save all the pictures to the hard disk is outlined below:

```
#Opent the set and jump to the first timestep
OpenDatabase("<path to your *.visit file>/files.visit", 0)

AddPlot("Volume", "energy")
p = VolumeAttributes()

#RayCast option
p.rendererType = 2
SetPlotOptions(p)
DrawPlots()

# Set the save window attributes
s = SaveWindowAttributes()
s.format = s.JPEG
s.progressive = 1
s.fileName = "test"
s.saveTiled = 1
SetSaveWindowAttributes(s)

#save every frame
for state in range(TimeSliderGetNStates()):
    SetTimeSliderState(state)
    SaveWindow()
```

### 3.3.2 Simple tracer movie

This one opens a *.trace file and renders the tracer temperature for different time steps. It renderes only every 6th step.

```
OpenDatabase("../output/sndd000.trace")

#Add a plot and make it pretty
AddPlot("Pseudocolor", "tracer_temperature")
p = PseudocolorAttributes()
p.opacity = 0.25
SetPlotOptions(p)
DrawPlots()

# Set the save window attributes
s = SaveWindowAttributes()
s.format = s.JPEG
s.progressive = 1
```

```
s.fileName = "test"
s.saveTiled = 1
SetSaveWindowAttributes(s)

# Render pics and save them
for state in range(TimeSliderGetNStates())[::6]:
    SetTimeSliderState(state)
    SaveWindow()
```

# Chapter 4

# Creating a plug-in

To create a plug-in from scratch you could start by reading the Section about *Database plug-ins* in *How to get data into VisIt*.

## 4.1 Code skeleton

This section covers only information not available in the *GettingDataIntoVisit* manual. The basic idea is to create a VTK object and pass it to the computing engine. A VTK array is in *Fortran order* (the first index is the fastest varying). Start by creating a code skeleton with the helper *xmledit*. For help

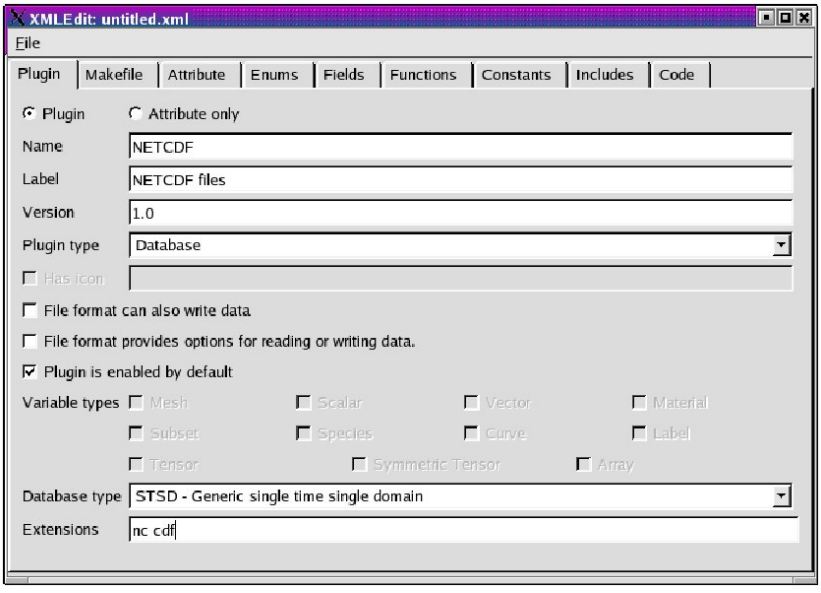


Figure 4.1: XMLEdit with some suitable entries

how to deal with the *xml* file created consult Section 3.2 of *Creating a database reader plug-in* in the *Getting Data into VisIt* manual. To import external libraries, add the files to *MDServer Files* and *Engine Files*. Don't forget to add

your `avt<plugin>FileFormat.C` as well, since the program will erase it from MDServer Files and Engine Files upon checking the buttons!

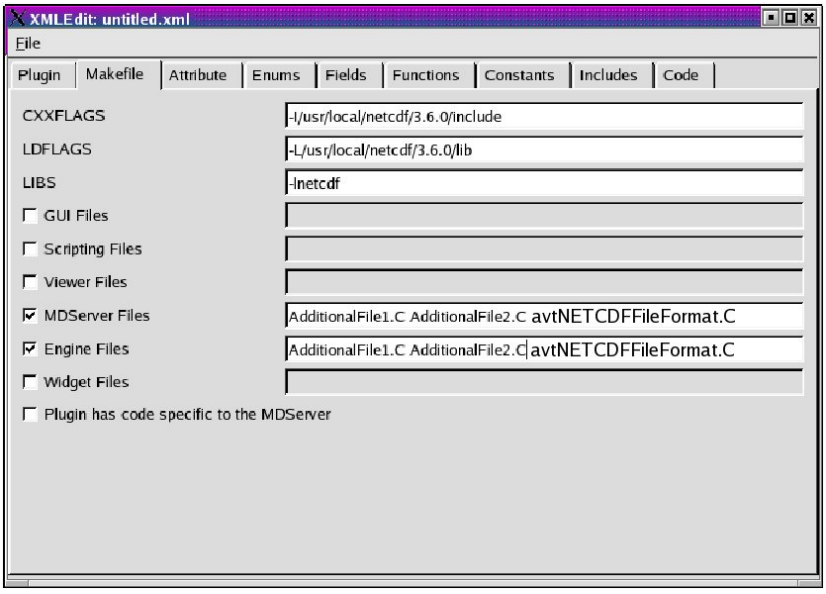


Figure 4.2: Don't forget to add the avn...FileFormat.C in MDServer Files and Engine Files!

## 4.2 The body

In order to get your plug-in running edit

```
avn...FileFormat.C
avn...FileFormat.h
```

and then *make* or *cmake . & make* your plug-in, depending on the version of VisIt. Open a file in the constructor of your plug-in class and read in the data as VTK arrays in the methods

```
PopulateDatabaseMetaData
GetMesh
GetVar
GetVectorVar
```

You can have a look at in the source code of many plug-ins at the Source Code Documentation.

- `http://www.visitusers.org/visit/1.10.0/doxygen/`

## 4.3 Debugging

You can start VisIt with debugging logs turned on. The logs will be written to the current directory, so make sure you have the permission to do so. For

further information about debugging have a look at Section 4.2 of *Creating a database reader plug-in* in the *Getting Data into VisIt* manual.
In VisIt 1.12 there is also the possibility to start VisIt using *Valgrind* (a tool for memory leak detection).

```
./visit -valgrind --track-origins=yes
```

However, I found it was easier for me to split my code first into parts running independently from VisIt and then Valgrinding them.

## 4.4 Using the visit-plugin tool for packing the plug-in

The `visit_plugin` script can also create the tar.gz file for distribution which used in Section 2.1. To do so clean the plug-in directory from all unnecessary files

```
make clean
rm foo.C bar.C~ ...
```

then invoke the script in the parent directory

```
cd ..
visit_plugin -p <pluginnane>
```

If you need help do

```
visit_plugin
```