Tutorial by Felicia Brisc, Universität Hamburg

A. Requirements for the time dimension in your NetCDF data sets

ParaView's **NetCDF Reader** follows the **CF Conventions**. For the time axis to be recognized correctly, your dataset must include a time dimension with the following mandatory attributes:

- standard_name = "time"
- units in relative time format (e.g., "days since 1900-01-01", "seconds since 1970-01-01 00:00:00", etc.)

Absolute dates (e.g., units = "YYYY-MM-DD HH:MM:SS") are not supported by the reader and will prevent the time dimension from being detected.In the example below, the mandatory time dimension and its attributes are highlighted in magenta:

```
netcdf myfile {
dimensions:
        time = UNLIMITED ; // (6 currently)
        x = 600 ;
        y = 600 ;
variables:
        double time(time) ;
                time:standard_name = "time" ;
                time:units = "years since 1-1-1 00:00:00" ;
                time:calendar = "proleptic_gregorian" ;
                time:axis = "T" ;
...}
```

B. How to use the provided Python script in ParaView's Programmable Filter

Provided files:

- **NetCDF_TimeRemapper.py**, the Python script

- The time file: **PARAVIEW_DATETIME_LIST_BCE.txt** file. This file contains the new custom time values in the -4000-01-01T00:00:00 format. There are 6 time steps

- Two NetCDF example files with six time steps each: **surf_fw_out_pv_tut.nc** and **pism_ANT_pv_tut.nc**

Important note on the tutorial example:
The displayed value range of the variables, and the chosen color maps are used **only for illustration purposes and to make temporal changes clearly visible**. This is due to the very small magnitude of the values and the limited number of time steps in the tutorial dataset. These choices have no specific scientific or physical meaning.

1. Open in any text editor the **PARAVIEW_DATETIME_LIST_BCE.txt** file and have a look at the new custom time steps values.

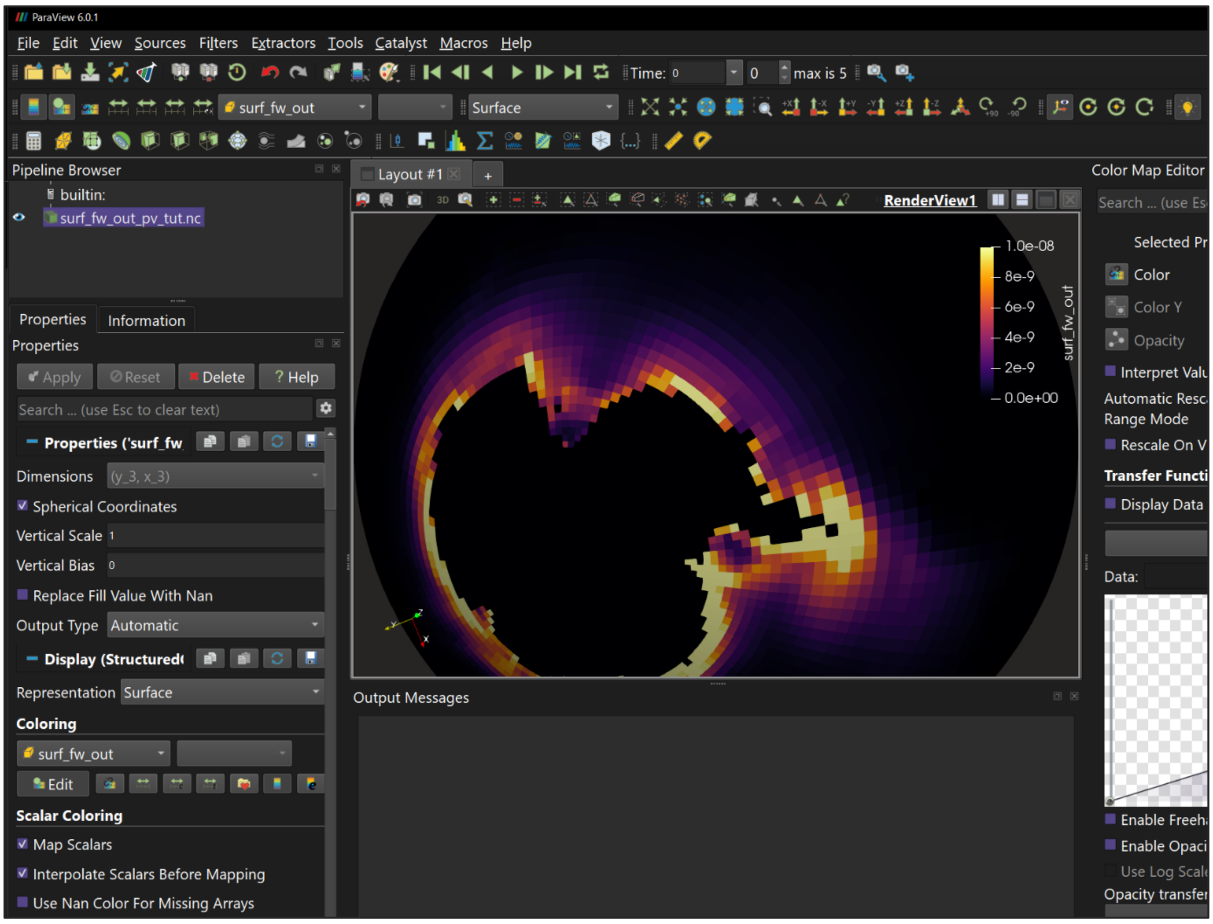
2. Open the Python script **NetCDF_TimeRemapper.py** in your text editor.
You will notice that the script is divided into three clearly marked sections.
In the first section (labeled “1. RequestInformation Script”), find the line that contains the path to the file **PARAVIEW_DATETIME_LIST_BCE.txt**.
Update this path so it points to the actual location where you saved that file on your computer.

3. Launch ParaView.

4. Open in ParaView the tutorial file **surf_fw_out_pv_tut.nc**.

File → **Open** → navigate to where the file is located → select “**NetCDF Reader**” in the “**Open Data With ...**” dialog window → click “**Apply**”

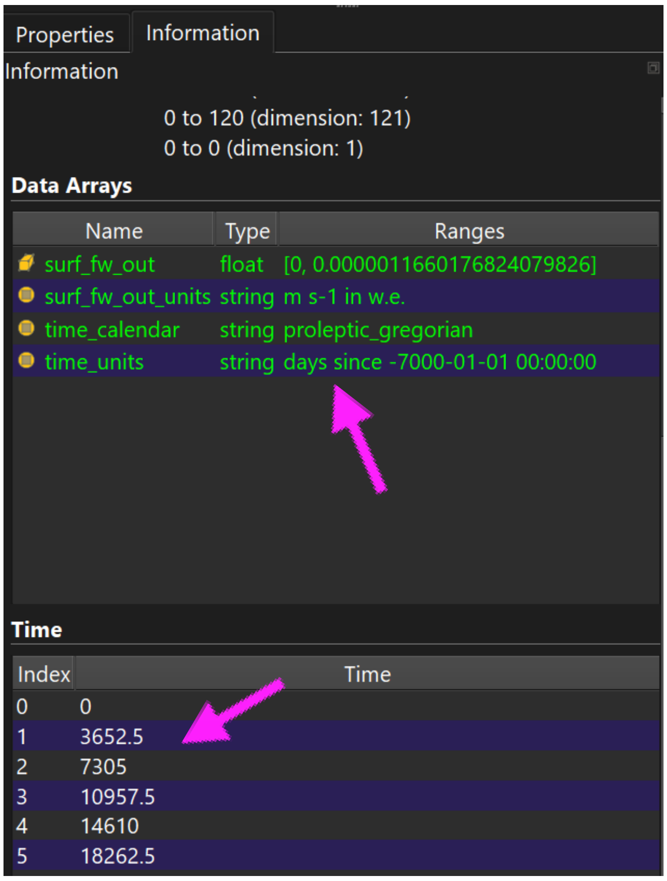
Visualize the **surf_fw_out** variable: select a color map and a value domain in the Color Editor. In the screenshot below (Figure 1) the value domain is very small: 0 … 1e-08 and the color map chosen was Inferno(matplotlib) from ParaView’s default color maps.



Properties ('surf_fw
Dimensions (y_3, x_3)
☑ Spherical Coordinates
Vertical Scale 1
Vertical Bias 0
☐ Replace Fill Value With Nan
Output Type Automatic
Display (Structured
Representation Surface
Coloring
surf_fw_out
Scalar Coloring
☑ Map Scalars
☑ Interpolate Scalars Before Mapping
☐ Use Nan Color For Missing Arrays

Figure 1

If we look in the **Information** tab we will see the variable **surf_fw_out**, the **time dimension attributes and its six values** (Figure 2). Play from the animation controls the time steps forward and backward to see the evolution of the variable.



Properties | Information

Information

0 to 120 (dimension: 121)
0 to 0 (dimension: 1)

Data Arrays

Name	Type	Ranges
surf_fw_out	float	[0, 0.0000011660176824079826]
surf_fw_out_units	string	m s-1 in w.e.
time_calendar	string	proleptic_gregorian
time_units	string	days since -7000-01-01 00:00:00

Time

Index	Time
0	0
1	3652.5
2	7305
3	10957.5
4	14610
5	18262.5

Figure 2

5. Apply the Programmable Filter

First select our file in the **Pipeline Browser**, then apply a **Programmable Filter**: **Filters → Search →** type **Programmable Filter**, then select it (Figure 3). The **Programmable Filter** will appear in the **Pipeline Browser** (Figure 4).

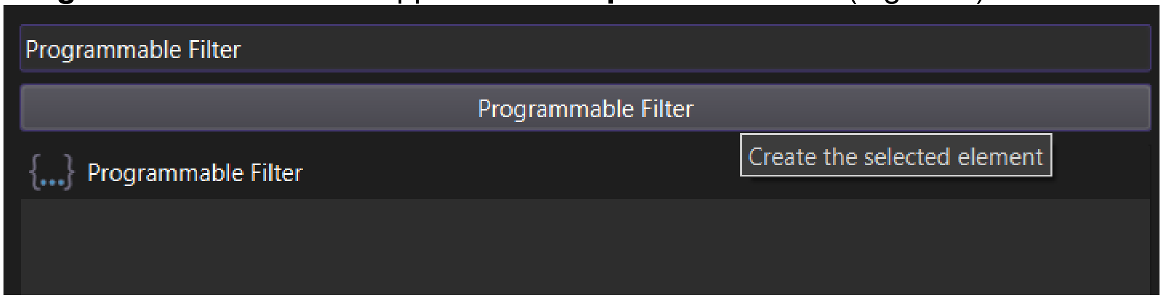


Figure 3

In the **Properties** tab of the **Programmable Filter**, you will see three scriptable boxes (Figure 4). These boxes allow you to provide Python code that hooks into three specific VTK/ParaView pipeline requests, which are executed automatically when the dataset is loaded and whenever time steps are navigated (forward, backward, or jumped to).

The three sections of the script are executed in the following order:

- RequestInformation Script
- RequestUpdateExtent Script
- Script (aka RequestData)

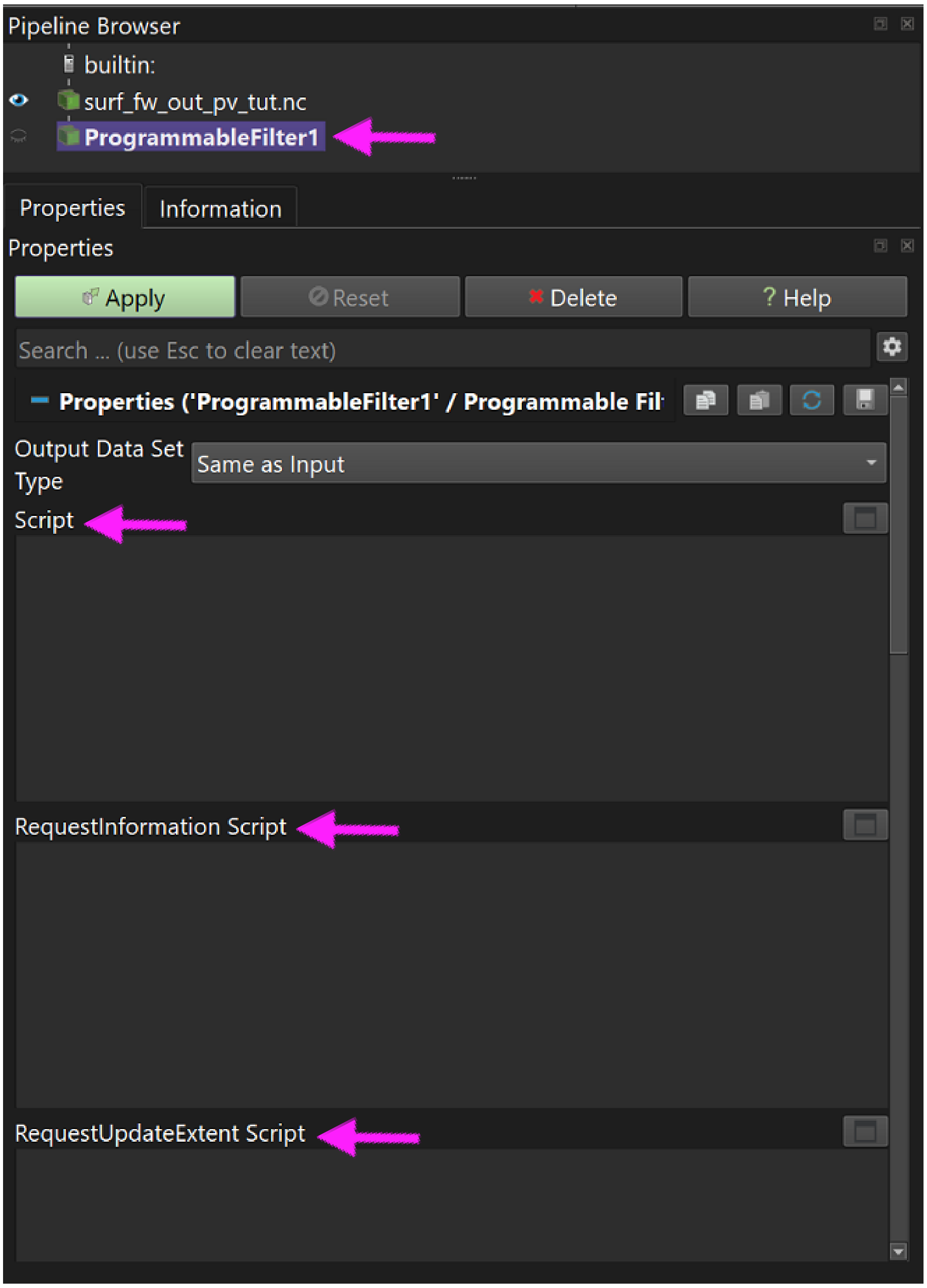
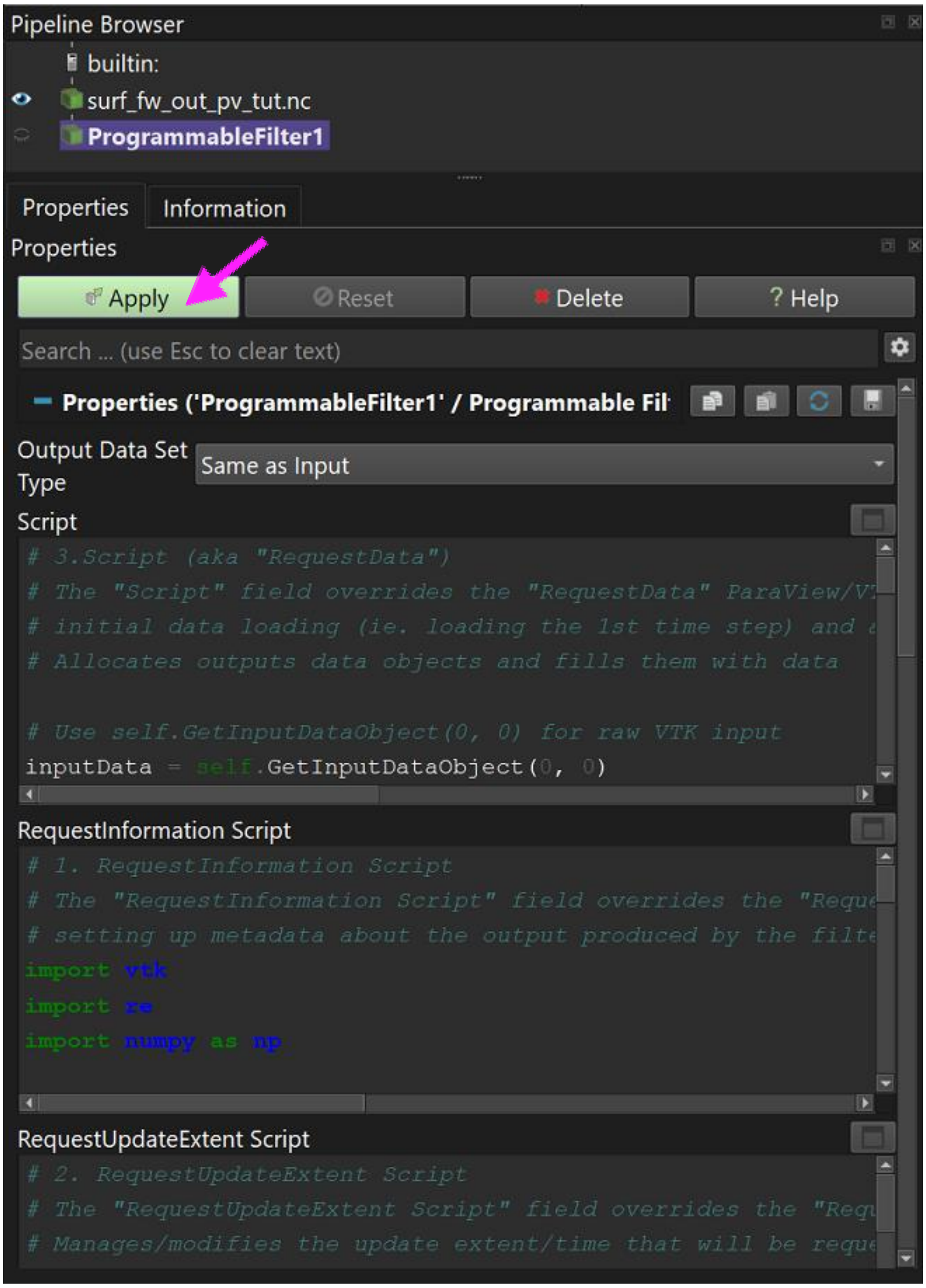


Figure 4

If you now look at the already-open **NetCDF_TimeRemapper.py** file in your text editor (**Step 1**), you will see that it is divided into three clearly marked sections, each

corresponding to one of the three boxes in the Programmable Filter.
Copy and paste each section into its matching box.
<u>After all three sections are in place</u>, click **Apply** in the **Properties** panel (Fig. 5).



Output Data Set Type: Same as Input

Script

```
# 3.Script (aka "RequestData")
# The "Script" field overrides the "RequestData" ParaView/V
# initial data loading (ie. loading the 1st time step) and
# Allocates outputs data objects and fills them with data

# Use self.GetInputDataObject(0, 0) for raw VTK input
inputData = self.GetInputDataObject(0, 0)
```

RequestInformation Script

```
# 1. RequestInformation Script
# The "RequestInformation Script" field overrides the "Reque
# setting up metadata about the output produced by the filte
import vtk
import re
import numpy as np
```

RequestUpdateExtent Script

```
# 2. RequestUpdateExtent Script
# The "RequestUpdateExtent Script" field overrides the "Requ
# Manages/modifies the update extent/time that will be reque
```

Figure 5

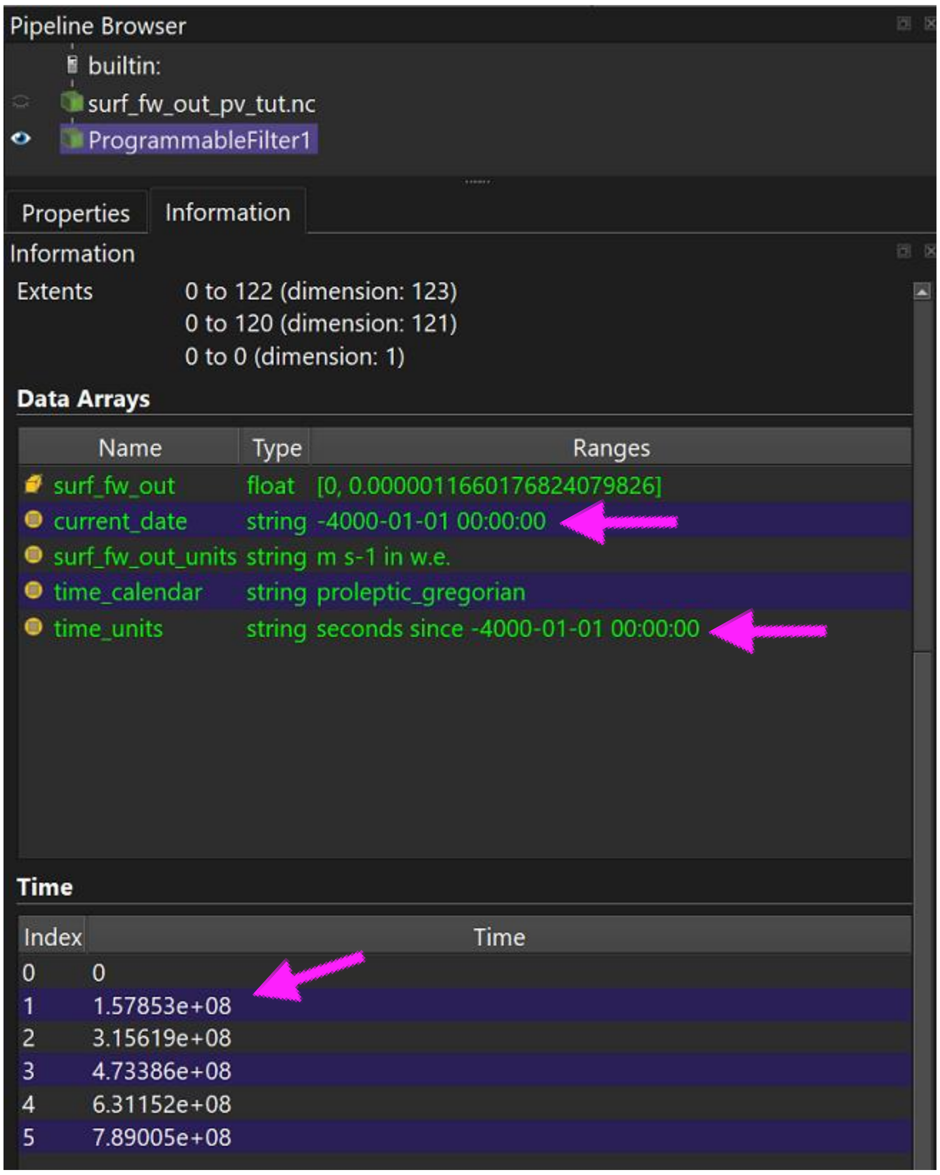
After clicking **Apply**, switch to the **Information** tab of the **Programmable Filter** (Figure 6). You will see that:

- The time step values are now taken from your text file and expressed in **seconds since the first date listed in that file**.

- The displayed time unit has changed to that first date (“seconds since -4000-01-01 00:00:00” in our case).

- A new array named **current_date** has appeared.

This **current_date** array was added by the script as an **variable (or attribute)** to the data set to make it easy for you to display the actual date and time as an annotation in the 3D view.



Pipeline Browser
- builtin:
- surf_fw_out_pv_tut.nc
- ProgrammableFilter1

Information

Extents	0 to 122 (dimension: 123)
	0 to 120 (dimension: 121)
	0 to 0 (dimension: 1)

Data Arrays

Name	Type	Ranges
surf_fw_out	float	[0, 0.0000011660176824079826]
current_date	string	-4000-01-01 00:00:00
surf_fw_out_units	string	m s-1 in w.e.
time_calendar	string	proleptic_gregorian
time_units	string	seconds since -4000-01-01 00:00:00

Time

Index	Time
0	0
1	1.57853e+08
2	3.15619e+08
3	4.73386e+08
4	6.31152e+08
5	7.89005e+08

Figure 6

Highly recommended: Save your work as a ParaView state file now!
Saving the state preserves the fully configured **Programmable Filter**, so in future projects you can simply load the state (then copy-paste the filter) and avoid rebuilding everything from scratch. **File → Save State**… (choose a meaningful name like **NetCDF_TimeMapper_Template.pvsm**)

6. (Optional) Display the date and time in the 3D viewport

Select our **Programmable Filter** → **Filters** → **Search** → type "**Annotate Attribute Data**" → select it (Figure 7)

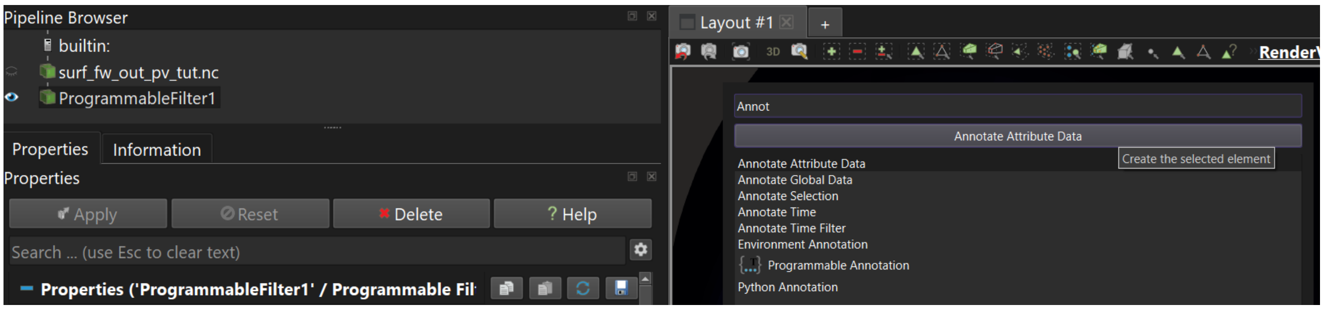
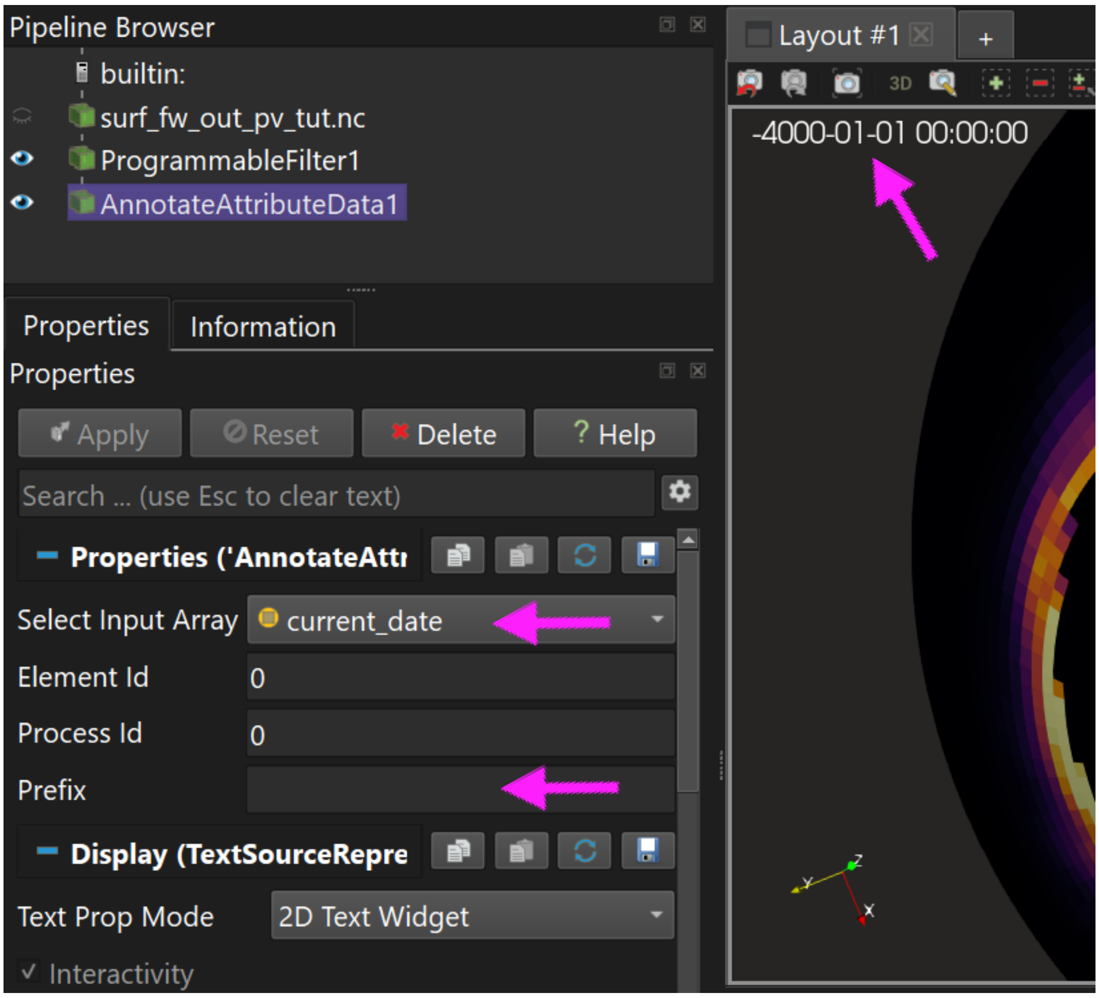


Figure 7

In the **Properties** tab, find and select the **current_date** variable in the **Select Input Array** dropdown list box. Delete any text in the **Prefix** field (leave it completely empty). Click **Apply**. The current date and time will now appear as interactive 2D text in the viewport. You can drag it to any position, change the font, color, size, or bold setting exactly like any other **Text** source in ParaView (Figure 8).



Select Input Array: current_date
Element Id: 0
Process Id: 0
Prefix:
Text Prop Mode: 2D Text Widget

-4000-01-01 00:00:00

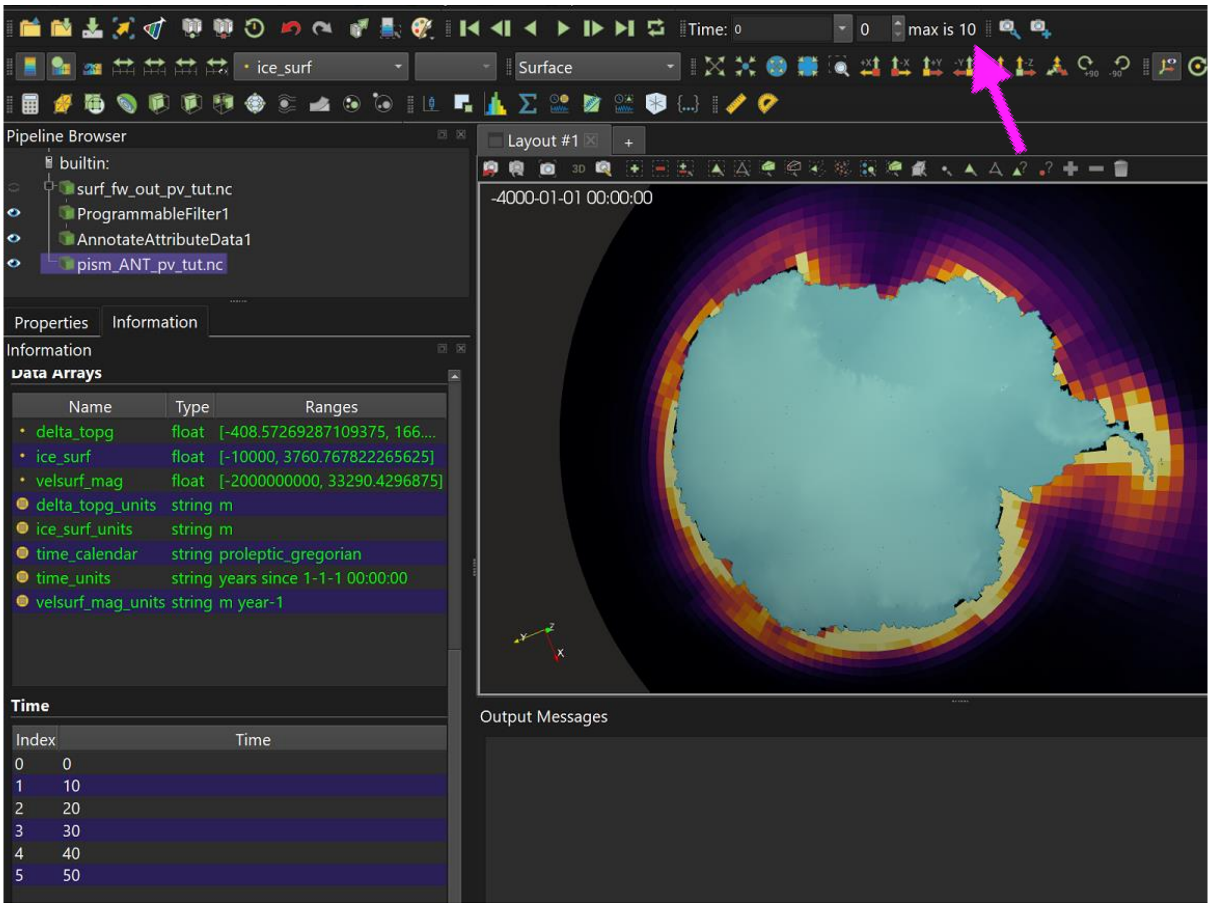
Figure 8

8. Reusing the Programmable Filter on a second dataset

Open the file **pism_ANT_pv_tut.nc** and visualize the **ice_surf** variable (Figure 9).

Go to the **Information** tab: you will notice that the time units and values are completely different from the first dataset (**surf_fw_out_pv_tut.nc**), and the number of time steps is now 10 instead of 5 in the animation time line.

If you play the animation, ParaView will first play all time steps of **pism_ANT_pv_tut.nc**, then all steps of **surf_fw_out_pv_tut.nc**.
This happens because ParaView uses a single global timeline, and the two files currently have incompatible time values - they cannot be animated in sync.



Pipeline Browser
builtin:
surf_fw_out_pv_tut.nc
ProgrammableFilter1
AnnotateAttributeData1
pism_ANT_pv_tut.nc

Properties | Information

Information

Data Arrays

Name	Type	Ranges
delta_topg	float	[-408.57269287109375, 166....
ice_surf	float	[-10000, 3760.767822265625]
velsurf_mag	float	[-2000000000, 33290.4296875]
delta_topg_units	string	m
ice_surf_units	string	m
time_calendar	string	proleptic_gregorian
time_units	string	years since 1-1-1 00:00:00
velsurf_mag_units	string	m year-1

Time

Index	Time
0	0
1	10
2	20
3	30
4	40
5	50

Figure 9

In order to remedy this, we will reuse the already-configured Programmable Filter:

Apply a new **Programmable Filter** to **pism_ANT_pv_tut.nc** - but do not click **Apply** yet ! (Figure 10)

Instead of copying the Python code again, simply reuse the filter you already set up:

Right-click the **first Programmable Filter** (the one under **surf_fw_out_pv_tut.nc**) → click on **Copy**.
Right-click the **new Programmable Filter** you just created → click on **Paste** to copy the Python content to this filter. Now click **Apply** on the second filter.

You will immediately see (Figure 13) that the total number of time steps in the animation bar drops back to 5 (ie. 6 time steps, 0…5). Both datasets now share exactly the same time values, so they animate perfectly in sync.

This copy-paste method is a fast way to apply the same time-mapping to additional NetCDF files.

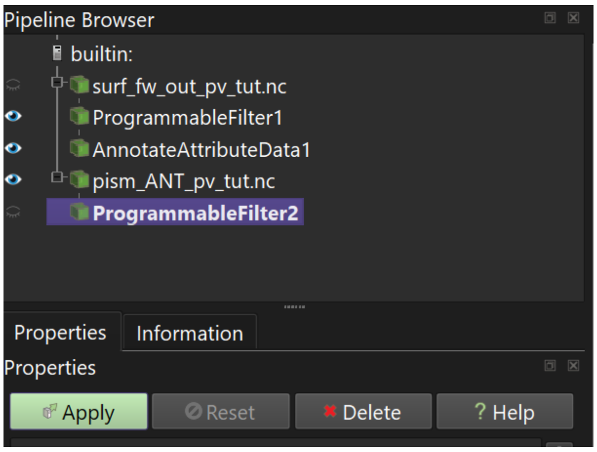


Figure 10

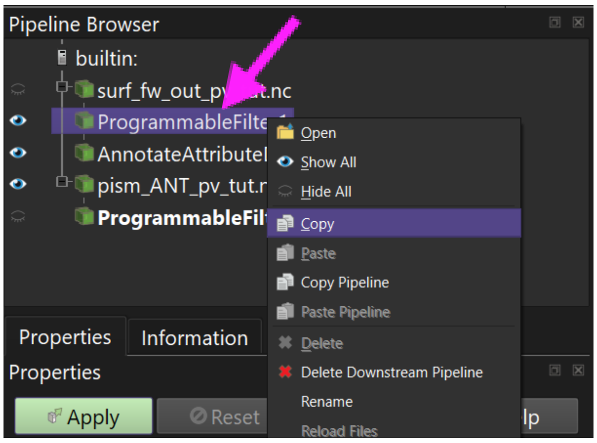


Figure 11

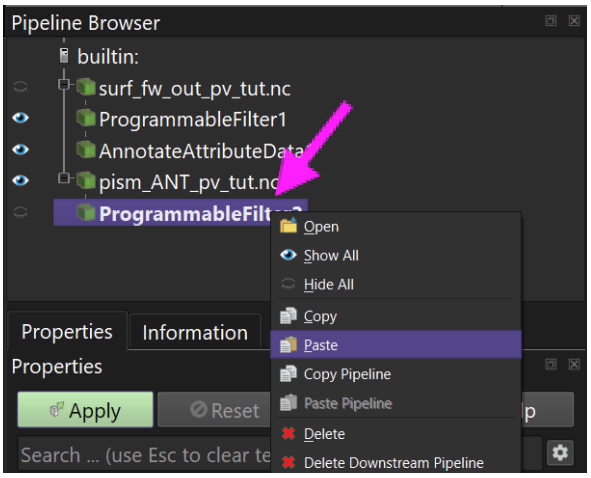
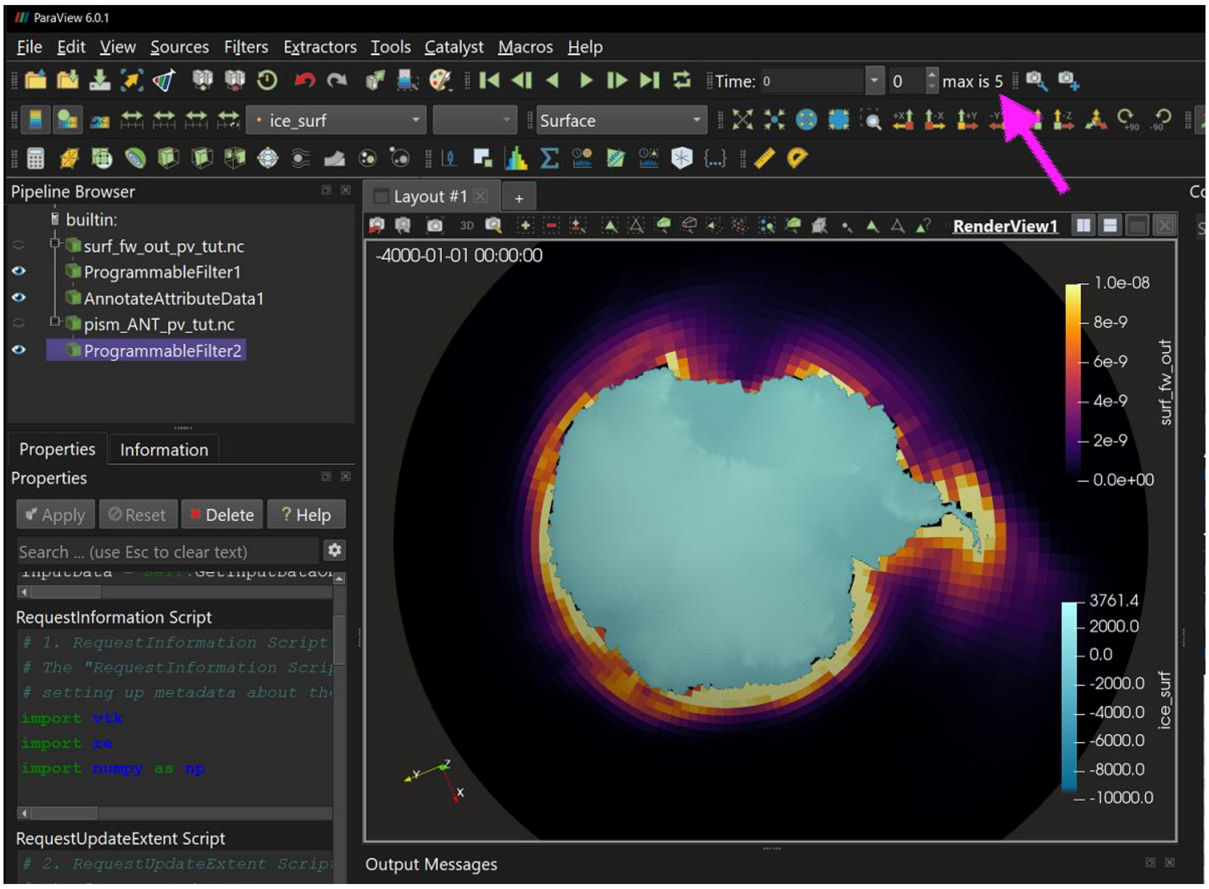


Figure 12



RequestInformation Script

```
# 1. RequestInformation Script
# The "RequestInformation Scri
# setting up metadata about th
import vtk
import re
import numpy as np
```

RequestUpdateExtent Script

```
# 2. RequestUpdateExtent Scrip
```

Figure 13

C. Important notes: Current limitations of reusing the Filter via Custom Filters or Plugins

ParaView offers two “user-friendly” ways to reuse a **Programmable Filter**, but neither currently works as expected for global time-step replacement:

1. Custom Filter / Save as Custom Filter

Right-click the working **Programmable Filter** → **Create Custom Filter** → give it a name → click **Finish** (or click through all the options options first, adjusting as needed). The new filter then appears under the **Filters** menu and can be applied with a single click.
However, in my experience - after applying this as a **Custom Filter**, the script loses direct access to the main VTK/ParaView pipeline. Time steps are modified only locally on the dataset itself, while the global animation timeline remains untouched. As a result, time steps from multiple files simply get appended instead of being synchronized.

2. Python Plugin

You can rebuild and package the script as a ParaView plugin (loaded via **Tools** → **Manage Plugins**). It will then appear permanently in the **Filters** menu.
Again, in my experience: It exhibits the same limitation as the **Custom Filter** – no global timeline control.

Conclusion and recommended workflow (most reliable for now):

1. Open a NetCDF data set, apply a **Programmable Filter** and build it as described in this tutorial.

2. Save the project (**File → Save State**…) as a reference project (e.g., **NetCDF_TimeMapper_Template.pvsm**).

3. When you wish to apply the Filter to a new NetCDF file, open this template project, open the NetCDF file from with this project, apply a **Programmable Filter** on it, then simply copy/paste the "template "configured **Programmable Filter** onto the new dataset as described in **§B, step 8**.

For now, copying and pasting the configured **Programmable Filter** remains the fastest and most robust reuse method. As future work, I hope to look deeper into the pipeline-access issue with Custom Filters and plugins – perhaps a future update will make fully reusable versions possible.