

KAMELEON

CONVERSION ● ACCESS ● INTERPOLATION

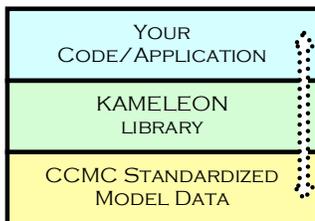
KAMELEON is a software suite that is being developed at the CCMC to address the difficulty in analyzing and disseminating the varying output formats of space weather model data. Through the employment of a comprehensive data format standardization methodology, Kameleon allows heterogeneous model output to be stored uniformly in a common science data format. The converted files contain both the original model output as well as additional metadata elements to create platform independent and self-descriptive data files. To facilitate model data dissemination, data reuse, and code reuse – the Kameleon access and interpolation library provides direct access to both the model data as well as the embedded metadata.

Features

- The interpolation routines can extract data directly from the disk or - for more data rich applications such as field line tracing - specified variables can be stored in main memory to reduce disk access and improve efficiency.
- Metadata extraction routines are provided to quickly extract both global and variable metadata elements imbedded in each standardized model output file.
- 4D interpolation - using the `time_interpolate()` routine, users can specify a directory of data, and extract data for any given variable, position, and time.
- The main library is written in C, yielding both static `.a` and shared `.so` libraries
- FORTRAN interface - for convenience, a fortran interface has been provided addressing fortran-to-c argument passing, string handling, and other compatibility issues.
- IDL interface – contains routines that are callable directly from IDL in the same manner as native IDL routines through LINKIMAGE.

Access and Interpolation Library Overview

After the original model output has been converted into a standardized science data format with accompanying metadata elements, access to the data can be achieved by utilizing the Kameleon Access and Interpolation Library. The



library provides a user friendly interface to the standardized data, offering both spatial and temporal interpolation functionality along with several metadata extraction routines. By using the Kameleon library, the underlying format and storage structure is hidden from the end user masking the complexity of the native interface associated with the implemented science data format. At the same time, users still have the option to use the native science data format access tools and routines providing even greater programming flexibility. The routines provided in the Kameleon access/interpolation library can be called from any C compatible programming language, allowing users to analyze model data using their preferred application. Integrating the Kameleon library into an existing application can be done in an efficient manner alleviating the need to become familiar with the internal storage layout and structure of the data files. The standardized model output used in conjunction with the Kameleon access/interpolation library, facilitates both data sharing as well as software reuse, by allowing the integration of new model data analysis capabilities into existing or new software applications.

Current Standardized Model Output Availability

3D Magnetosphere: BATSRUS

3D Magnetosphere: OpenGGCM/UCLA-GGCM

Ionosphere: CTIP

Heliosphere: ENLIL

Solar: MAS (Under Development)

Science Data Format Output Options:

CDF 2.7

CDF 3.0

HDF5 (coming soon)

Call From Any C Compatible Programming Language

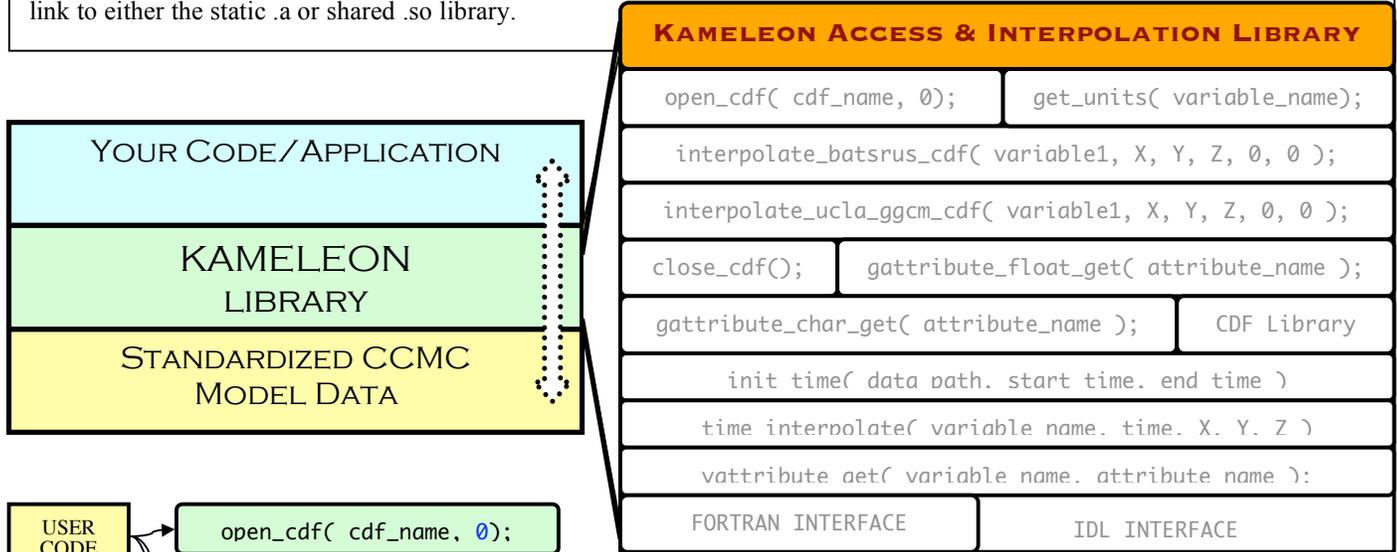
- Fortran
- Java
- C/C++
- Perl
- IDL
- Vtk
- OpenDx
- Your App

<http://ccmc.gsfc.nasa.gov>



Implementation

The Kameleon Access and Interpolation Library includes a set of example programs that illustrate how routines from the library can be implemented. C and FORTRAN example programs are included that perform spatial and temporal interpolation, as well as metadata extraction functions. Below are two of the example programs included with the distribution. The C example, `time_interp_example.c`, illustrates how 4D interpolation can be implemented from a simple C driver program. The FORTRAN example, `f2c_interp_open_ggcm.f`, outlines how spatial interpolation can be performed from a simple FORTRAN 77 driver program. The C and FORTRAN example programs are automatically compiled during the library build, and can be used out of the box. The Source code of the example programs can be used as guidelines on how to integrate specific library functionality into an existing application. Compile examples are also included describing how to compile and link to either the static `.a` or shared `.so` library.



```

USER CODE
  open_cdf( cdf_name, 0);
  interpolate_batsrus_cdf( variable1, X, Y, Z, 0, 0 );
  close_cdf();

program f2c_interp_open_ggcm
c   Three functions used to interpolate
c   data from a specified batsrus cdf file
  external f2c_open_cdf, f2c_close_cdf,
f2c_interp_bats_cdf
c   Variables to be used for interpolation and
data extraction
  character*150 cdf_file_path
  real*8 x,y,z
  real*8 interpolated_value
  integer status
  character*50 var_to_read
c   --- set your actual path name here ---
  cdf_file_path='open_ggcm.cdf '
c   Open the cdf file
  status=f2c_open_cdf(cdf_file_path)
c   --- set your position values in GSE ---
  x=-55.0
  y=12.0
  z=20.0
c   --- set name of variable of interest ---
  var_to_read='bx '
c   --- call the interpolation routine ---
  status=
1f2c_interp_open_ggcm_cdf(x,y,z,interpolated_valu
e,var_to_read)
c   --- close the currently open cdf file
  status=f2c_close_cdf(0)
  write(*,*) var_to_read, interpolated_value
end
  
```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main( int argc, char *argv[] )
{
  extern long init_time(char *, double *, double *);
  extern float
  time_interpolate(char*,double,float,float,float);
  long status;
  char data_path[750];
  char variable[10];
  float X, Y, Z;
  double time, start_time, end_time;
  float sample_time_interval;
  float time_interpolated_value;
  strcpy( data_path, argv[1] );
  strcpy( variable, argv[2] );
  X = atof( argv[3] );
  Y = atof( argv[4] );
  Z = atof( argv[5] );
  sample_time_interval = atof( argv[6] );
  status = init_time( data_path, &start_time, &end_time );
  printf("Simulation start_time:\t%f msec\n", start_time
);
  printf("Simulation end_time:\t%f msec\n", end_time );
  for( time=start_time;time<=end_time;time+=sample_time_int
erval)
  {
    time_interpolated_value=time_interpolate(variable,time,X
,Y,Z);
    printf( "%s [ %f, %f, %f ] @ %f
milliseconds\t%f\n",variable,
X,Y,Z,time,time_interpolated_value );
  }
  return 1;
}
  
```

For spatial interpolation, an example of how only three calls are required to get a value for any specified variable and position.

Highlighted Code – Kameleon Library Calls