*Astronomical Data Analysis Software and Systems XVII* **P8.13**
*ASP Conference Series, Vol. 394, © 2008*
*R. W. Argyle, P. S. Bunclark, and J. R. Lewis, eds.*

# The Transform Library – A High-Level Interface to Coordinate Systems

Janine Lyn, Douglas Burke, Mark Cresitello-Dittmar, Stephen Doe, Ian Evans, Janet DePonte Evans, Gregg Germain, Jonathan McDowell, Joseph Miller

*Smithsonian Astrophysical Observatory, 60 Garden Street, M.S. 67, Cambridge, MA, 02138, USA*

**Abstract.** The Transform Library is a new, stand-alone software package developed by the Chandra X-ray Center (CXC), that provides a convenient high-level C++ interface for performing World Coordinate System transformations. The library wraps a subset of the lower-level wcslib functions to provide an easy interface to both users and developers. The Transform library is designed to be used within C++ programs and various scripting environments. Notably, it provides high-level user interfaces in Python and S-Lang for ease of use.

The Transform Library consists of C++ classes and methods for performing transformations on input base arrays (table data or image axes), for accessing and manipulating required transform parameters, and for calculating the transform matrix. Transform types include pixel to world coordinate transforms and vice versa, as well as linear transforms and scaling transforms. The design allows for transform chaining, so the user is able to combine multiple transforms into more complex arrangements.

The Transform Library will be integrated in the new versions of ChIPS and Sherpa that will be released in CIAO4. In addition, CIAO users will be able to use the library directly via the scripting languages. This provides the ability to easily create highly specialized applications to suit the user's particular needs.

## 1. Introduction

The Transform Library was designed as an independent library which would provide an easy way of storing and manipulating transforms and their parameters, and a transparent way of computing the transformed values of virtual data columns and image axes. It is currently being used in Crates, the CXCDS I/O interface to the DataModel, and has been incorporated into Sherpa, a spectral and spatial fitting package, and ChIPs, a plotting package.

Figure 1 illustrates the Transform Library's design architecture and its relationship to other libraries and interfaces. The Transform Library is composed of a C++ library and higher-level user interfaces in Python and S-Lang. These main components are discussed in the following sections.

## 2. The Transform Library C++ Layer

The C++ library consists of a base Transform class with several derived classes which extend the transform definition and provide the same user interface for all Transforms. Each Transform sub-class already knows what parameters it

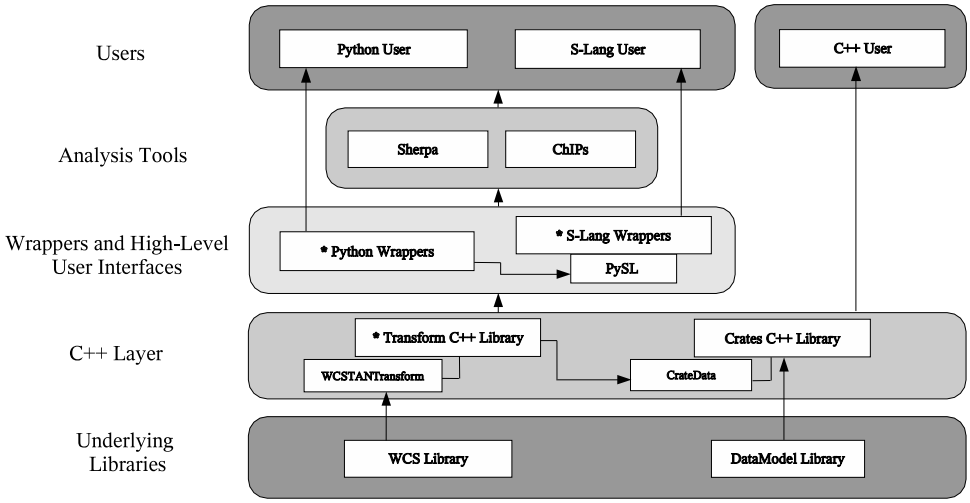


Figure 1. An Overview of the Design Architecture. The starred (*) boxes indicate the main components of the Transform Library.

needs, the dimensionality and datatype of the input data, and how to perform the calculation.

The Transform class, in order to be more generally usable, applies the transform to a base array of data of a particular datatype and returns the transformed values in the same datatype. If possible, the class can also perform the inverse transformation on an input array. The derived classes also contain overloaded apply and inverse methods to accommodate most datatypes.

The Transform metadata are stored in a simple class called TransParam. All Transform sub-classes have methods for querying the Transform for a list of its parameters. The resulting parameter list only requires user-input parameter values.

Table 1 contains a brief description of each of the Transform sub-classes and their required parameters.

## 3. The Controller Transforms

Additionally, the C++ Library contains two Controller Transforms which extend the Transform class and are responsible for managing the execution of a group of transforms.

- SERIAL Transforms let the user arrange member transforms to be executed sequentially.
- PARALLEL Transforms apply their member transforms concurrently.

These Transforms allow the user to create complex reusable arrangements with numerous Transforms while facilitating data handling and transform application.

Table 2 presents a step-by-step example of how to use the Controller Transforms to implement a Polynomial Transform as represented in Figure 2.

Table 1. The Transform Sub-Classes

| Sub-Class | Action | Parameters |
|---|---|---|
| ADD | add elements of two inputs | none |
| LINEAR | linear scale | SLOPE, INTERCEPT |
| LINEAR2D | apply 2D linear transform | SCALE, OFFSET, ROTATION |
| MULT | multiply elements of two inputs | none |
| POLY | apply polynomial transform | CONST_A, CONST_B, OFFSET |
| POW | raise to power | EXPONENT |
| SCALE | multiply by a factor | FACTOR |
| SQR | square | none |
| SUM | sum all inputs | none |
| WCSTAN | apply WCS transform | CRPIX, CRVAL, CDELT, CROTA |
| SERIAL | (see Section 3) | |
| PARALLEL | (see Section 3) | |

Table 2. Sample code for building a Polynomial Transform. The user has the POLY Transform class available and should not need to create one by hand.

Transform: result = $x^2$ + ax where x is a N element array and a is a constant

```
// Create a SQR transform with name = 't1'
Transform* t1 = new SQRTransform('t1');

// Create a SCALE transform with name = 't2'
Transform* t2 = new SCALETransform('t2');

// Retrieve the FACTOR parameter and input the scale factor for t2
TransParam* params = t2→get_parameter('FACTOR');
params→set_value( a );

// Create a PARALLEL transform to execute t1 and t2 concurrently
Transform* t3 = new PARALLELTransform(t1, t2);

//Create an ADD transform
Transform* t4 = new ADDTransform();

// Create a SERIAL transform to add the resulting arrays of t3 and t4
Transform* t5 = new SERIALTransform(t3, t4);

// Apply the t5 transform to the x array
result = t5→apply( x );
```

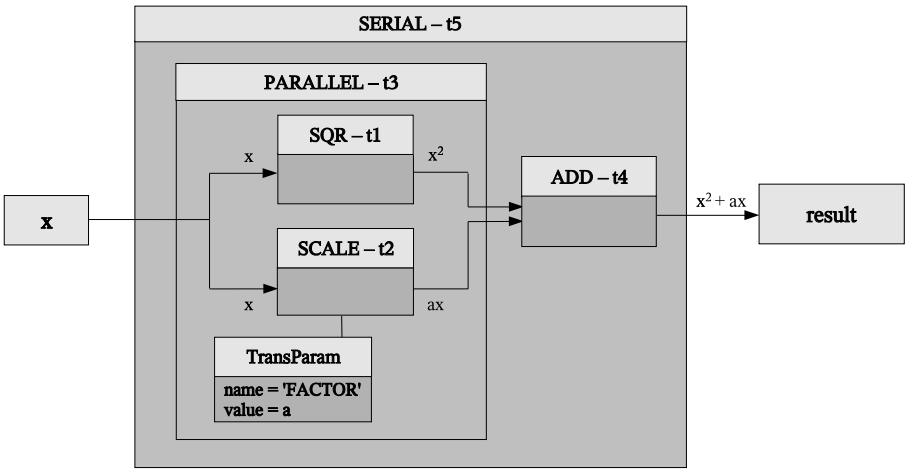


Figure 2. A graphical representation of the data-flow inside the composite Polynomial Transform: result = $x^2$ + ax.

## 4. Wrappers and High-Level User Interfaces

The Transform Library is extensible to other scripting environments. By using SWIG (Simplified Wrapper and Interface Generator) to create wrappers in Python to interface with the C++ layer, all Transform classes and most methods are accessible through the Python wrappers.

The Library also provides several higher-level methods for use in both Python and S-Lang (via PySL and the S-Lang wrappers):

- apply_inverse_transform
- apply_transform
- get_axis_transform
- get_transform
- get_transform_matrix
- print_axis_names
- print_transform
- set_transform
- set_transform_matrix

**Acknowledgements.** Support for the development of the Transform Library is provided by the National Aeronautics and Space Administration through the Chandra X-ray Center, which is operated by the Smithsonian Astrophysical Observatory for and on behalf of the National Aeronautics and Space Administration contract NAS8-03060.