Why Marry the ROOT and .NET Worlds?

I am much more productive in languages like python and C# than I am in C++. Many of the functional features (lambdas, Language Integrating Query (LINQ)) make these languages much more concise in expression problems. In addition, the recent activity around functional-like languages (msamlc, etc.) on the .NET platform are bringing a new way of expressing problems in very powerful forms rather than our usual imperative forms.

.NET (and the Java platform as well) use a runtime and allow trivial sharing of libraries between languages, making the library available for one language (C#) generally means it is available for all other languages (e.g., .NET).NET was chosen over the JVM because of the features most important to me (JavaBeans, LINQ) were not available in other tools.

.NET is the de facto data storage and histogramming, and fitting, platform for the field of HEP. This project is the infrastructure for an approach to marry these two worlds.

Initialization is driven by a number of small .NET developers, located at the University of Washington.

This project is the infrastructure for an approach to marry these two worlds. It is likely there are other bugs lurking.

Dynamic Interface Library

D# version 4.0 the dynamic keyword was included. This allows for late-binding: not knowing what object you are operating on at the time of compilation. This is exactly what is used by python, and the pyROOT introduction takes advantage of this. Even the case of the strongly typed wrappers described here, dynamic wrappers prove useful in two cases if an object unknown by the wrappers must be operated on, or if one wishes to have only a minimal library.

Microsoft's DLR (Dynamic Language Runtime) is capable of many optimizations. The performance gain of the dynamic interface used by ROOT.NET is as insignificant as it can get. For example, most dynamic languages are slower than C++, but they are very dynamic—very dynamic and nearly the same amount of testing that the fully typed static wrappers must be operated on, or if one wishes to have only a minimal library.

The current implementation is horrible, and it involves many object allocations on each call to the .NET runtime. This is exactly what is used by python, and the pyROOT introduction takes advantage of this. Even the case of the strongly typed wrappers described here, dynamic wrappers prove useful in two cases.

For global variables define a class static variable. In .NET does not allow static variables at the global scope, so to access them through the class.

For enums generate a .NET enum that directly has the values of the C++ enum.

For classes, define a wrapper class with a pointer to the real ROOT class, and for each method translate the arguments to C++, call the method, and then possibly translate the result back.

For classes, define a wrapper class with a pointer to the real ROOT class, and for each method translate the arguments to C++, call the method, and then possibly translate the result back.

For classes, define a wrapper class with a pointer to the real ROOT class, and for each method translate the arguments to C++, call the method, and then possibly translate the result back.

For classes, define a wrapper class with a pointer to the real ROOT class, and for each method translate the arguments to C++, call the method, and then possibly translate the result back.

For classes, define a wrapper class with a pointer to the real ROOT class, and for each method translate the arguments to C++, call the method, and then possibly translate the result back.

For classes, define a wrapper class with a pointer to the real ROOT class, and for each method translate the arguments to C++, call the method, and then possibly translate the result back.