Role of the Data Dictionary

LCG Persistency Workshop
5 June 2002
P. Mato / CERN

What it is?

- What we call *Data Dictionary* is also called:
  - Reflection
  - Run-type type information (RTTI)
  - Introspection
- Repository of all the transient "types" of the system and their description (at least the ones that we want to make persistent, interact with, etc.)
  - Programming language dependent
  - Several dictionaries may co-exist

Functionality

- Typical functionality
  - Get the type of an existing object
  - Dynamically create an instance of a type
  - Bind the type to an existing object, and then
    - » Invoke the type's methods or access its data members
- Additional functionality
  - Create new types at run-type
  - Populate dictionary from other dictionaries

Uses of the Data Dictionary

- Type/Data Browsers
  - Display type and display/modify object information
- Scripting
  - Invoking methods
- Data Persistency
  - Saving and restoring object state (e.g. serialization)
- Distributed Applications
  - Sending and receiving objects through the network (e.g. XML+SOAP, serialization)

Programming Languages

- C++
  - Provides very limited RTTI functionality
- Java
  - Reflective interface is part of the language/library
  - Only the "read" one is available

  (*Class, Field, Method, Constructor,Array, Modifier,…*)
- C#
  - Built-in reflection
  - Both "read" and "write" interfaces are available

  (*Type, Assembly, Module, FieldInfo, MethodInfo, …*)

  (*TypeBuilder, MethodBuilder, FieldBuilder, …*)

Dictionary Interface(s)

- Reflective Interface
  - Interface to obtain the “meta” information, access to data members, and invocation of methods.
  - Main interface for dictionary “clients”
- Emit Interface
  - Construct the dictionary in memory when loading libraries
  - Possibility to build new types at run-time

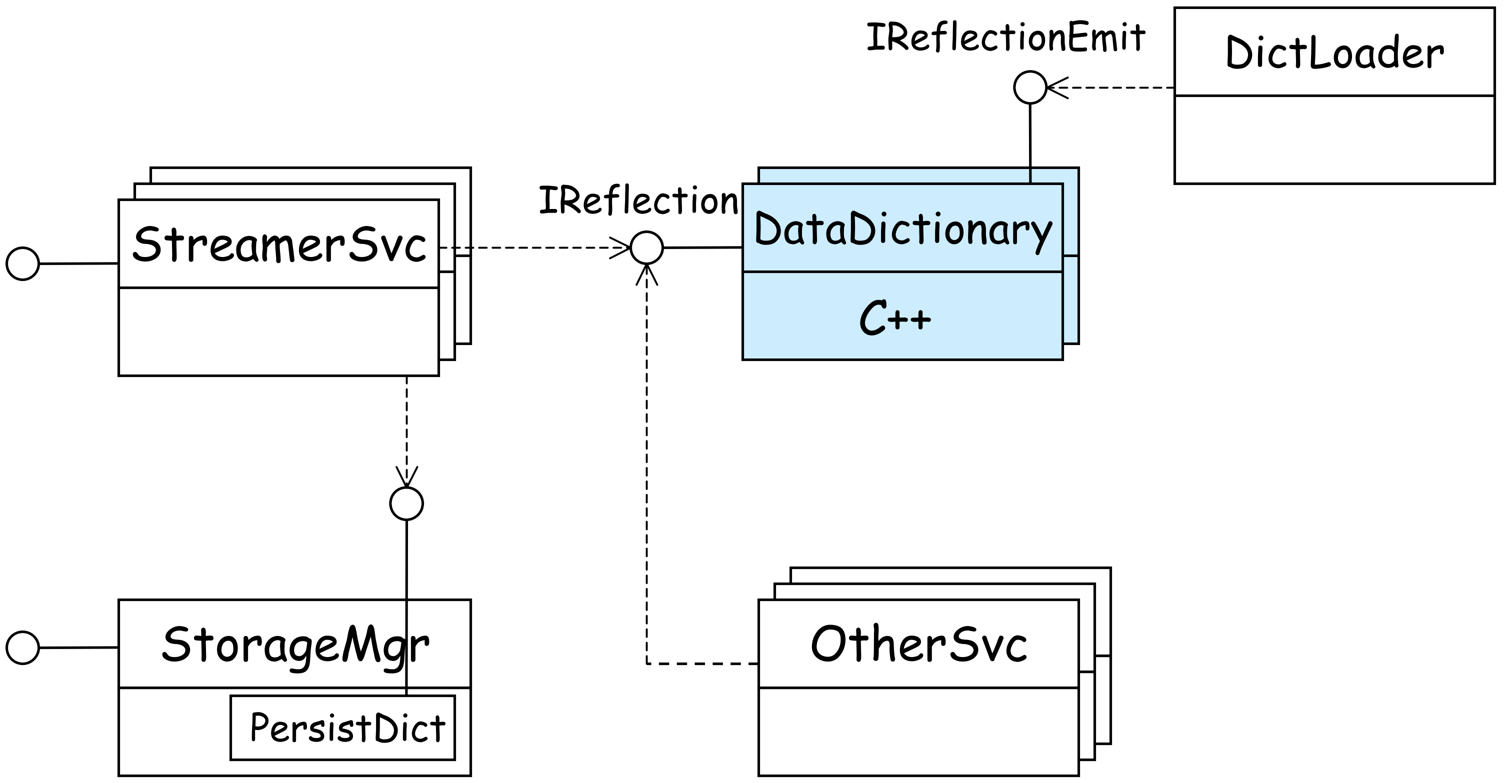
Dictionary Requirements

- What should the data dictionary contain?
  - All standard C++ features
    - » Namespaces
    - » Classes, inheritance
    - » References and pointers
    - » Data members and methods with arguments
    - » Templates
    - » … extra information needed by clients (e.g. description, persistency flags)
- The interface depends of programming language
  - Not always the case (e.g. C#, VB.Net,…)

Interface Choice

- Provide a Reflective interface to C++ as it would have been provided by the language itself
  - Get inspiration from existing interfaces (Java, C#,…)
    - » Easy of use, intuitive
    - » Completeness
- Neutral and self-contained
  - Should not be bound to any "framework", "scripting language", etc.

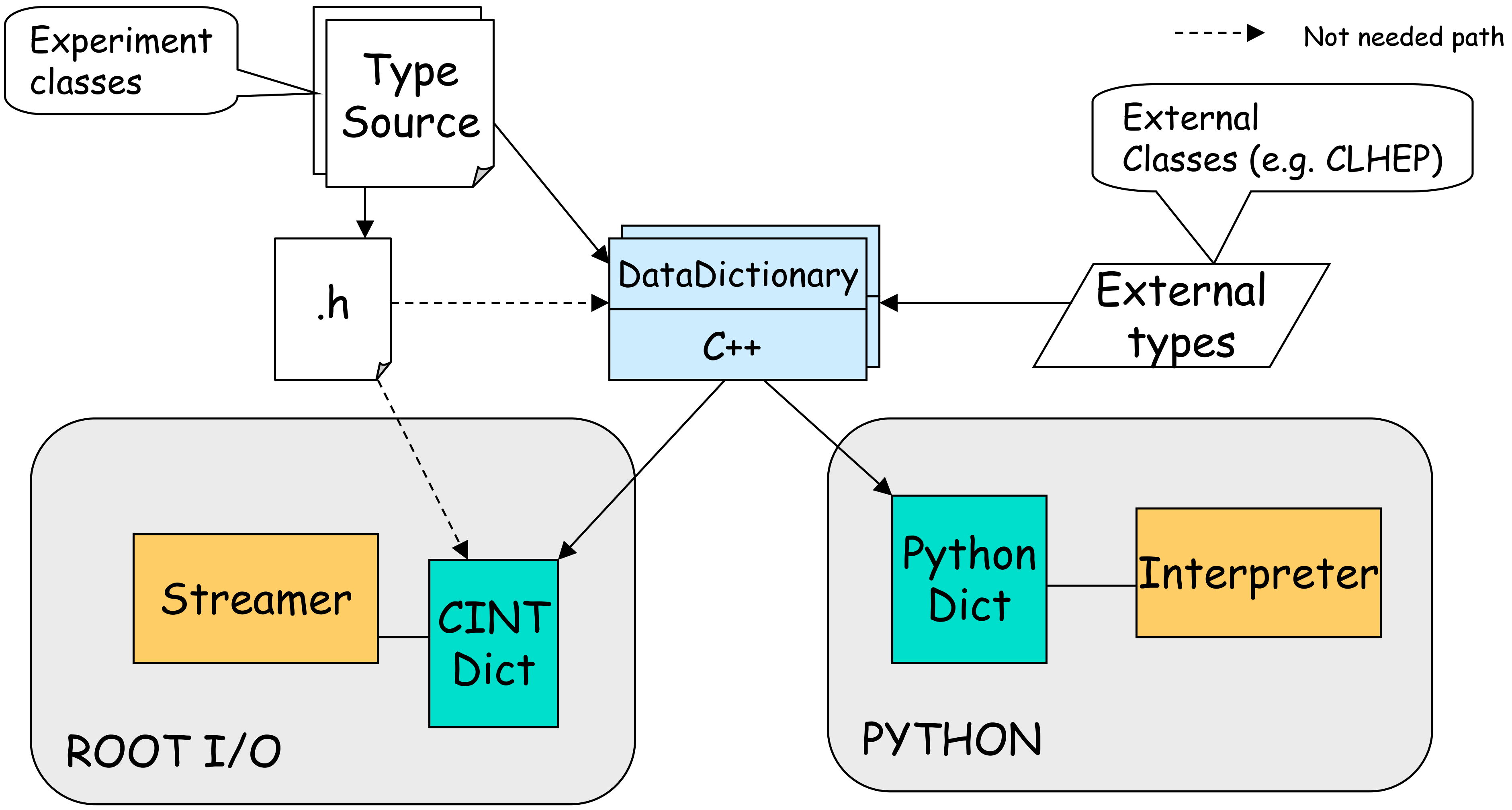
Fitting in the Persistency Framework



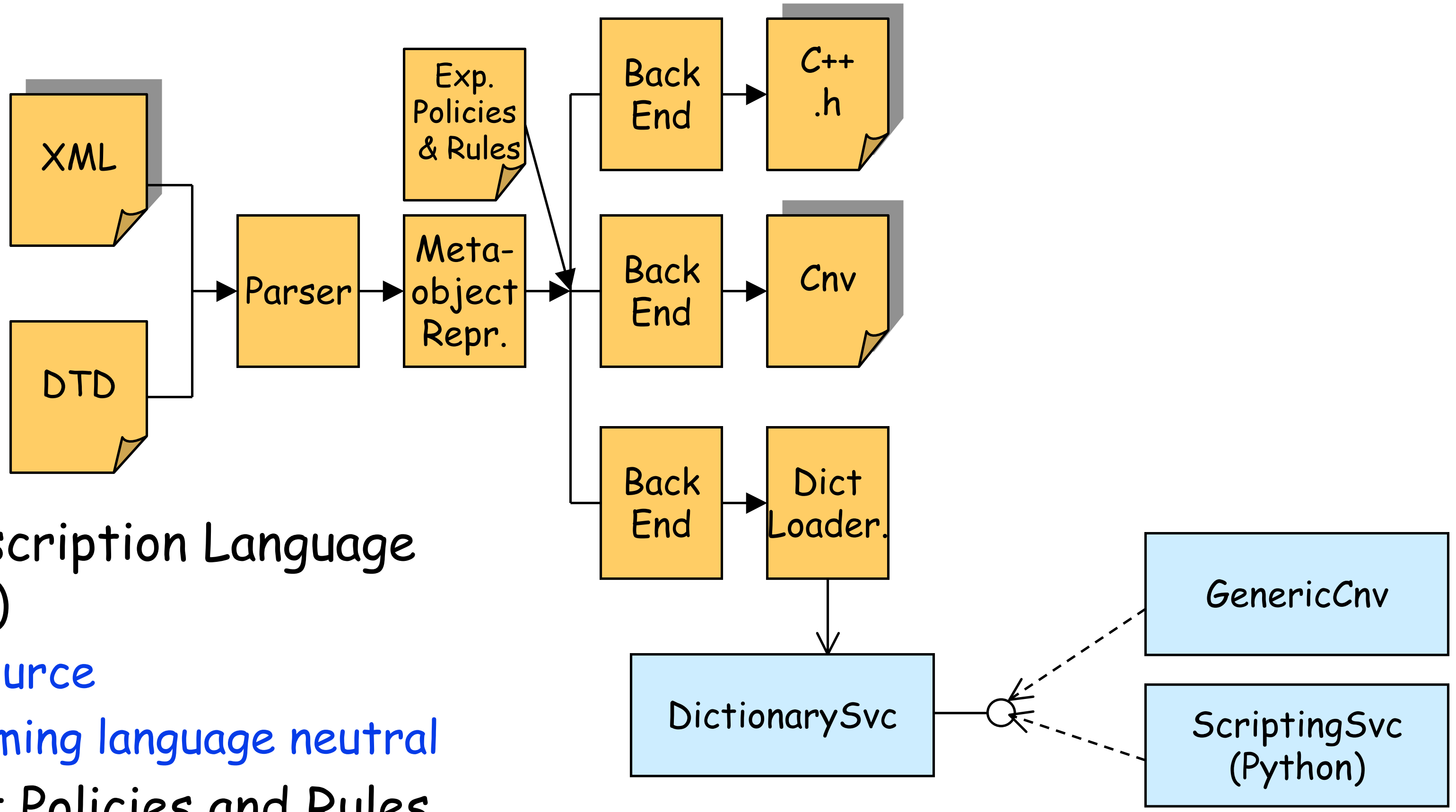
Filling the Dictionary

- Many different ways to provide the data dictionary information
  - The compiler should do it (not possible for C++)
  - Make use of the debug information (e.g. Espresso)
  - Parsing header files (e.g. CINT, SWIG)
  - Generating dictionary libraries from other sources (e.g. ATLAS/LHCb)
  - Writing by hand dictionary libraries (e.g. external classes)
  - …
- Filling dictionary "on-demand"
- **Goal**: fill a single **master** dictionary

ROOT I/O and Python examples



Object Description (LHCb/ATLAS)



- Object Description Language (XML, ADL)
  - Single source
  - Programming language neutral
- Experiment Policies and Rules
- Various back-ends

Summary

- Data Dictionary is needed for many reasons (persistency is only one)
- Would have been nice if C++ had a reflective interface to access the data dictionary
  - We have somehow to overcome the limitation
  - Borrow ideas from more modern languages
- I tried to convince you:
  - Dictionary independent of Persistency/Scripting/…
  - Common LCG dictionary from which other framework specific dictionaries can be populated
  - The way to fill the dictionary should be the experiment choice