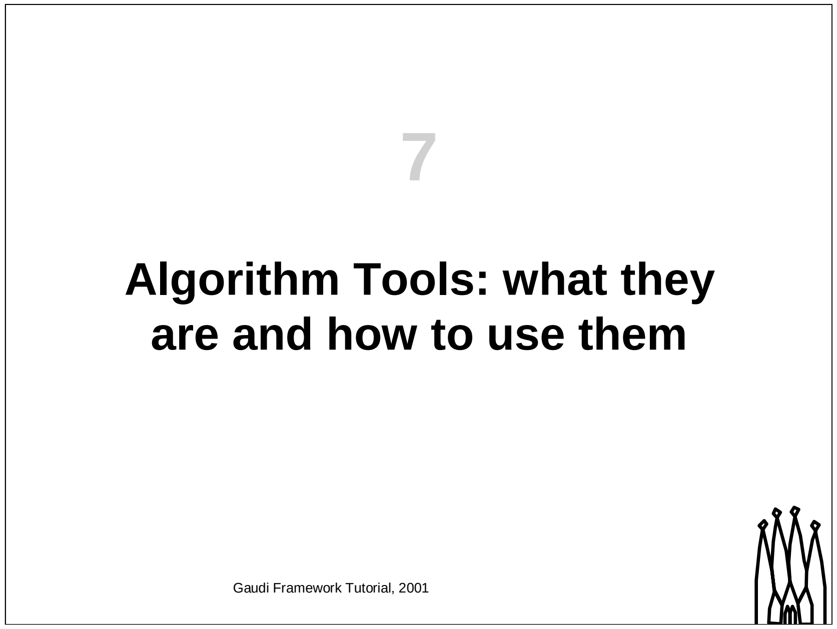


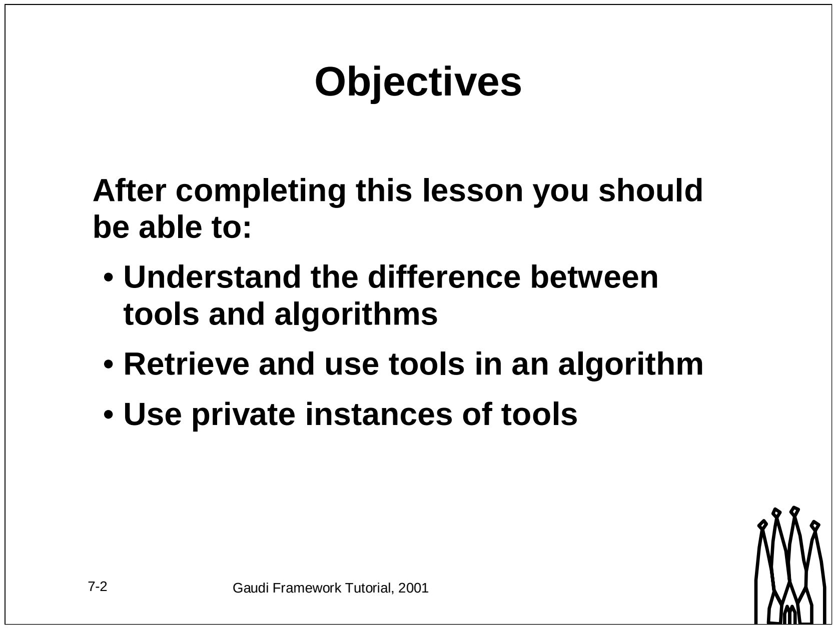
7

Algorithm Tools: what they are and how to use them

Gaudi Framework Tutorial, 2001

Schedule:

Timing	Topic
20 minutes	Lecture
30 minutes	Practice
50 minutes	Total



Objectives

After completing this lesson you should be able to:

- **Understand the difference between tools and algorithms**
- **Retrieve and use tools in an algorithm**
- **Use private instances of tools**

7-2 Gaudi Framework Tutorial, 2001

Goals

The first goal of this lesson in Algorithm tools is to introduce this new concept. Understanding the differences between with other components of the framework (Services, Algorithms) is important for making consistent designs across the experiment.

The second objective of the lesson is to enable you to use tools, understanding the difference between public and private instances of tools.

Caveat

Things that will be used in the exercises and are assumed to be know:

- **How to write a simple algorithm accessing MCParticle & MCVertex**
- **How to use the Message Service**
- **How to use Job Options**

… essentially what you learned yesterday

Caveat:

The aim of this part of the tutorial is to learn how to use and write tools.

Some of the topics covered in the “Gaudi Basics Tutorial” are assumed to be known and are consequently used.

Why Algorithm Tools?

Sometimes in an Algorithm it is necessary to execute the same operation more than once per event, only for some events, on specific non identifiable data objects, producing new objects that will be put in the Event Store later

➔ Needed to introduce a new concept

The need for Algorithm Tools

When implementing an Algorithm, very often it is necessary to perform some operations or complex processing for each object in a list (e. tracks in the track collection). Therefore, the concept of sub-algorithm is not adequate for that purpose. In this kind of operations is more efficient to pass the data as arguments instead of registering and retrieving from the transient store. In addition, it can be that the same kind of operation can be re-used in other algorithms.

Algorithm Tools

The concept of Algorithms Tools, being a sort of simple algorithms callable many times and with arguments, was introduced in Gaudi

- **Examples**
  - **Vertexing**
  - **Track transport**
  - **Association to truth**
  - **Filtering of particles based on a pID CL**

Designing for Re-use

Algorithm Tools are useful small algorithms that can be packages in a way that will be easy to re-use them in other Algorithms. They are callable many times during the execution of an event and the user can pass arguments.

Examples:

- Vertexing (to produce one or many vertexes from a list of tracks or particle candidates)
- Track transport (to obtain the track parameters on other points of the detector)
- Association to truth (to obtain the Monte Calo information corresponding to a reconstructed object)
- Selection from a container of objects (to reduce the a list ob objects according to some selection criteria)

Requirements

- **An Algorithm requires a Tool on a per need base**
- **The ToolSvc checks if the requested type of Tool is available and returns an instance of the requested type after configuring it**
- **An Algorithm can have a private instance of a Tool that can be configured according to its need via JobOptions**
- **An Algorithm can use a Tool “as-it-is”, in this case the ToolSvc will provide a “common” Tool (shared-Tool)**
- **An Algorithm should be able to use a Tool without worrying about the implementation**
- **The ToolSvc keeps track of the Tools instantiated and if a tool exists it does not re-create it**
- **An Algorithm can tell the ToolSvc if it will not need a tool anymore**
- **The ToolSvc will decide which tools instances can be deleted (always done at finalization of the Service)**
- **Not only Algorithms would want to use Tools (also Services …)**

ToolSvc

The ToolSvc is the service that manages Algorithm Tools (private or shared)

- **Algorithms ask the ToolSvc for a given Tool by name, it can keep a pointer to a tool and use it when necessary**
- **Manages the life-cycle of Tools creating them on a first request basis**
- **Keeps track of existing Tools, holding all instances and dispatching them as requested**

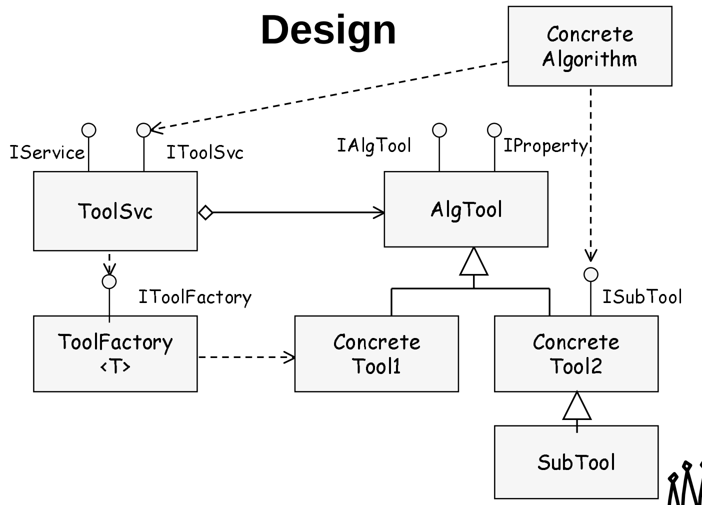
Tools can be configured using the JobOptions as Algorithms or Services

The ToolSvc Service

This service is managing Algorithm Tools. It is the service in charge of tools in their life-cycle, it creates them on first request, configures them and destroys them at the finalize phase of the job.

An Algorithm requests the ToolSvc to obtain a reference to a Tool by its name. Tools can be private or shared. The idea is that if a Tool requires a fair amount of resources (memory, cpu time for configuration) it does make sense to share the Tool among the various Algorithms that may require this functionality. The problem with a shared tool is that is can not keep a state between invocations since it is not guaranteed that other Algorithms may have used it meanwhile.

Design

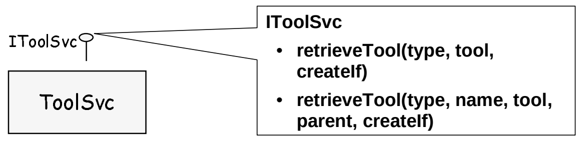


Design

The Design follows closely that of Algorithms. We have a base class (AlgTool) from which all the Tools inherits from. This base class ensures that the Tool is manageable and configurable from the TooSvc service and JobOptions service. Concrete Tools will have their own interface specific to the tool or class of tools. Of course, specializations can always be possible by inheritance of concrete tools.

The ToolSvc uses the factory pattern to instantiate concrete tools without known about them using the generic abstract interface IToolFactory which is implemented by the template class ToolFactory. The service keeps the list of AlgTools existing in the program.

How to retrieve a tool via the ToolSvc



IToolSvc

- **retrieveTool(type, tool, createIf)**
- **retrieveTool(type, name, tool, parent, createIf)**

In order to retrieve a tool it is necessary to specify its concrete class and a pointer return type

```
IMyTool* pMyTool = 0;
retrieveTool("MyToolClass", pMyTool)
```

Note that the ToolSvc interacts with the tools interfaces

Configuring a tool

A concrete tool can be configured using the jobOptions

Follow usual convention:

IdentifyingNameOfTool.NameOfProperty

```
ToolSvc.MCUtilityTool.PrintDepth = 0;
```

ParentName.ToolName

Through the base class all tools have the OutputLevel property

- The default value is that of the parent

Hands on: Use MCUtilityTool

Write a simple algorithm that retrieves and uses the MCUtilityTool to print a decay tree given a MCParticle

- Take the DecayTreeAlgorithm and use it as a starting point to make an AnalysisAlgorithm
- Request the tool to the ToolSvc using the tool interface (will talk more about Tools interfaces later)
- Replace the printDecay method of the algorithm by using that of the tool
- Use the matchDecayTree method to check for a 1-step decay (ex. $J/\psi \rightarrow \mu^+\mu^-$)

You can find the IMCUtilityTool documentation at:
`http://lhcbsoft.web.cern.ch/LHCbSoft/Phys/DaVinci/v2r0/doc/html/class_i_m_c_utility_tool.html`

Hands on

In the following exercise we will use a simple Monte Carlo utility tool.

Printing a MonteCarlo tree given the parent particle is in fact something that many algorithms could want to do: this functionality could be in a tool. At the same time many algorithms could want to check if a specified decay is in the event.

Hands on: Using IToolSvc to retrieve a tool

```
...AnalysisAlgorithm.h
  class IMCUtilityTool*;                 ///< Forward declaration
  std::vector<std::string>  m_daugName; ///< Name of daughters found
  std::vector<long>         m_daugID;   ///< GeantID of daughters
  std::string        m_toolType;        ///< Tool type
  IMCUtilityTool*    m_pUtilTool;       ///< Reference to tool
```

The interface

```
...AnalysisAlgorithm::initialize()...
  sc = toolSvc()->retrieveTool( m_toolType, m_pUtilTool );
  if ( sc.isFailure() ) {
   // You have to handle the error!
  }
```

Tool Concrete class

Hands on: Using a tool

```
...AnalysisAlgorithm::execute()
// inside loop over MCParticle
if ( (*ipart)->particleID().id() == m_partID )   {
  log << MSG::INFO << "Found Particle of type " << m_partName
      << endreq;
  // Now use tool to print tree and check if requested tree
  m_pUtilTool->printDecayTree( 0, " |", *ipart );
  bool result = m_pUtilTool->matchDecayTree( *ipart,
                                             m_daugID );
  if ( result ) { // Print a message
  } else { // Print a different message
}
Don’t forget to include DaVinciTools/IMCUtilityTool.h
and GaudKernel/IToolSvc.h
```

7-13 Gaudi Framework Tutorial, 2001

Public and Private tools

- **A tool instance can be shared by many algorithms: it is public and belong to the ToolSvc**
- **Algorithm can have private instances of tools, that can be configured “ad-hoc”**
- **To retrieve a private instance the parent has to pass itself to the retrieve method**

```
toolSvc()->retrieveTool(“MyTool”,pMyTool, this)
```

7-14 Gaudi Framework Tutorial, 2001

Public and Private Tools

A tool instance can be shared by different algorithms and services. Its parent is the ToolSvc.

It is possible to re-configure such instances but care has to be taken to ensure no undesired side effects in the algorithms that use it.

In addition an algorithm could need to use two differently configured instances of the same tool

Private instances of tools address these points. Although they are managed by the ToolSvc, they are seen as belonging to the algorithm requesting them. The algorithm has to pass itself in order to notify the ToolSvc that the tool instance has to be private.

Hands on: Private Tools

Use two instances of the same tool type in the AnalysisAlgorithm

- **A public instance with default configuration**
  - When the specified decay tree is not found
- **A private instance with different printing depth**
  - When the specified decay tree is found

Note:

The tool is not modified, only its configuration via the job Options.

Hands on: Using a private and a public instance of a tool

```
...AnalysisAlgorithm.h
  std::string        m_myToolType;     ///< Tool type
  IMCUtilityTool*    m_pMyUtilTool;    ///< Reference to tool

...AnalysisAlgorithm::initialize()...
  sc = toolSvc()->retrieveTool( m_myToolType, m_pMyUtilTool,
                                this );
...AnalysisAlgorithm::execute()...
  m_pMyUtilTool->printDecayTree( 0, " |", *ipart );
  bool result = m_pMyUtilTool->matchDecayTree( *ipart,
                                               m_daugID );
  if ( result ) { … } else {
     m_pUtilTool->printDecayTree( 0, " |", *ipart );
  }
```