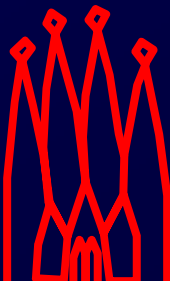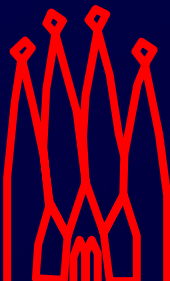# 3

Job Options and Printing

Job Options

- **All applications run the same main program (`gaudirun.py`)**

- **Job options define what to run, in what order, with what data, with what cuts**

- **Similar to "cards" files in old Fortran frameworks**

- **Accessed through IJobOptionsSvc interface**
  - **Details hidden from users by the framework**

Job Options: Data Types

Primitives

- bool, char, short, int, (long, long long), float, double, std::string
- And unsigned char, short, int, (long, long long)
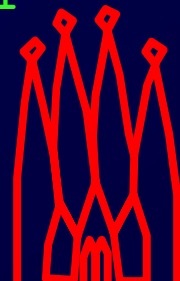
Vectors of primitives

- std::vector<bool>, std::vector<double>...

Pairs, Maps

- e.g. std::pair<int,int>, std::map<std::string,double>

The full list of possible types is documented at:
`http://cern.ch/proj-gaudi/releases/latest/doxygen/_parsers_8h.html`

CERN Tutorial, October 2009

Using Job Options

Declare property variable as data member

```
class MyFirstAlgorithm : public GaudiAlgorithm  {
private:
    double m_jPsiMassWin;

    ...
};
```
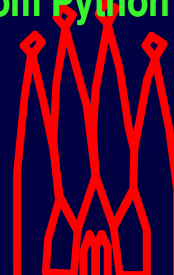
LHCb convention for member data

Declare the property in the Constructor, and initialize it with a default value

```
MyFirstAlgorithm::MyFirstAlgorithm( <args> )
{
    declareProperty( "MassWindow",

        m_jPsiMassWin = 0.5*Gaudi::Units::GeV,

        "The J/Psi mass window cut" );
}
```

Property name, used in job options file

Variable, initialized to default value

Documentation string, useful from Python

Setting Job Options

Set options in job options file

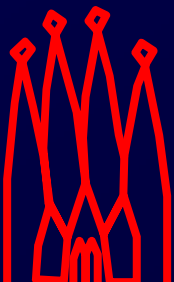- **File path(s) given as argument(s) of executable**

  ```
  gaudirun.py ../options/myJob.py [also .opts]
  ```

- **Python syntax**

  - Type checking
  - Expressions, if-then-else, loops etc.
  - Early Validation of configuration

- **Example**

  ```
  MyFirstAlgorithm("Alg1").MassWindow = 10.* GeV
  MyFirstAlgorithm("Alg2").MassWindow = 500. # Default is MeV

  Class("ObjectName").PropertyName = PropertyValue
  ```

Types of job options

- **Distinguish job options that organise sequencing of algorithms in LHCb applications:**

```
MCITDepCreator = MCSTDepositCreator("MCITDepCreator")
ApplicationMgr().TopAlg += [ MCITDepCreator ]
```

- **From options that change the behaviour of algorithms and tools:**

```
MCITDepCreator.tofVector = [25.9, 28.3, 30.5]

TheITTool = STSignalToNoiseTool( "STSignalToNoiseToolIT" )
TheITTool.conversionToADC = 0.0015;
```
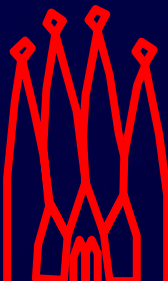
LHCb conventions for job options

- **Job options that organize the sequencing of algorithms are tagged and released with the application (typically in xxSys packages)**
  - These are increasingly replaced by "ConfigurableUser" classes (e.g. Brunel(), DaVinci()). See Configurables tutorial

- **Options that change the behaviour of algorithms and tools should be initialized to sensible defaults in the .cpp**
  - If needed, any options different from the defaults (e.g. if there are several instances of the same algorithm with different tunings) are taken from files stored in the corresponding component packages

```
importOptions("$STALGORITHMSROOT/options/itDigi.opts")
```

```
.opts files can be included ("imported") into python options
```
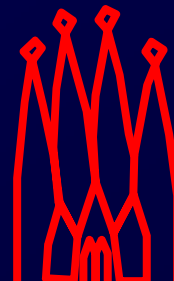
Job Options You Must Know

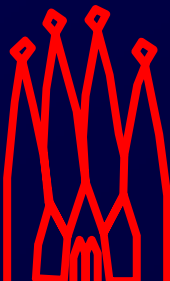`ApplicationMgr().EvtMax`   `<integer>`

Maximal number of events to execute

   (if -1 all events in input files)

`ApplicationMgr().TopAlg   <List of configurables>`

Top level algorithms to run: Type("Name")
   e.g.: [MyFirstAlgorithm("Alg1"), MyFirstAlgorithm("Alg2")]

Also defines the execution order

CERN Tutorial, October 2009

Printing

Why not use std::cout, std::cerr, ... ?

- **Yes, it prints, but**
  - **Do you always want to print to the log file?**
  - **How can you connect std::cout to the message window of an event display?**
  - **How can you add a timestamp to the messages?**
  - **You may want to switch on/off printing at several levels just for one given algorithm, service etc.**

CERN Tutorial, October 2009
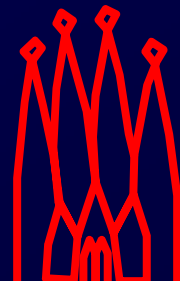
Printing - MsgStream

Using the MsgStream class

- **Usable like std::cout**

- **Allows for different levels of printing**
  - **MSG::VERBOSE**   **(=1)**
  - **MSG::DEBUG**   **(=2)**
  - **MSG::INFO**   **(=3)**
  - **MSG::WARNING**   **(=4)**
  - **MSG::ERROR**   **(=5)**
  - **MSG::FATAL**   **(=6)**
  - **MSG::ALWAYS**   **(=7)**

- **Record oriented**

- **Allows to define severity level per object instance**
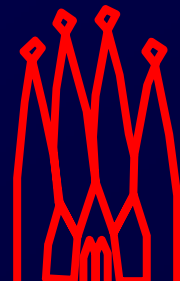
MsgStream - Usage

Send to predefined message stream

```
info() << "PDG particle ID of " << m_partName
        << " is " << m_partID << endmsg;

err() << "Cannot retrieve properties for particle "
      << m_partName << endmsg;
```
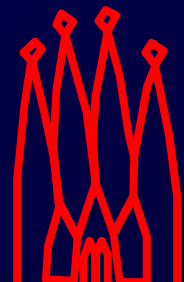
Print error and return bad status

```
return Error("Cannot retrieve particle properties");
```

Formatting with format("string", vars)

```
debug() << format("E: %8.3f GeV", energy ) << endmsg;
```

Set printlevel in job options

```
MessageSvc().OutputLevel      = ERROR
MySvc().OutputLevel           = WARNING
MyAlgorithm().OutputLevel     = INFO
```

Print everything of INFO level or higher

CERN Tutorial, October 2009

Units

We use Geant4/CLHEP system of units

– mm, MeV, ns are defined to have value 1.

– All other units defined relative to this

– In header file "GaudiKernel/SystemOfUnits.h"

– In namespace Gaudi::Units

Multiply by units to set value:

```
double m_jPsiMassWin = 0.5 * Gaudi::Units::GeV;
```

Divide by units to print value:

```
info() << "Mass window: " << m_jPsiMassWin / Gaudi::Units::MeV
       << " MeV" << endmsg;
```

## Units can be used also in job options:

```
import GaudiKernel.SystemOfUnits as Units
SomeAlgorithm().MassWindow = 0.3 * Units.GeV
```

StatusCode

- ## Object returned by many methods
  - Including GaudiAlgorithm::initialize(), GaudiAlgorithm::execute() , etc.

- ## Currently, takes two values:
  - StatusCode::SUCCESS
  - StatusCode::FAILURE

- ## Should always be tested
  - If function returns StatusCode, there must be a reason
  - Report failures:
    ```
    StatusCode sc = someFunctionCall();
    if ( sc.isFailure() )
       { Warning("there is a problem", sc ).ignore(); }
    ```

- ## If IAlgorithm methods return StatusCode::FAILURE, processing stops
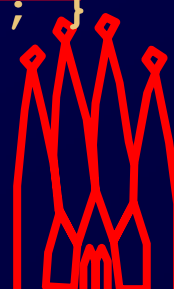
Exercise

Now read the web page attached to this lesson in the agenda and work through the exercise