3

Job Options and Printing

Job Options

- Job is steered by “cards” file
- Options are not directly accessed
- Access 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

Arrays of primitives

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

Associative properties:

- std::map<std::string,std::string>,…

The full list of possible types is available in `$GAUDIKERNELROOT/GaudiKernel/Parsers.h`

Using Job Options

Declare property variable as data member

```
class TutorialAlgorithm : public DVAlgorithm  {
private:
  double m_jPsiMassWin;
  ...
};
```

LHCb convention

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

```
TutorialAlgorithm:: TutorialAlgorithm( <args> )
<initialization>
{
  declareProperty( “MassWindow",
                m_jPsiMassWin = 0.5*Gaudi::Units::GeV );
}
```

Intialization to default value

Setting Job Options

- Set options in job options file
  - File path is first argument of executable

```
$DAVINCIROOT/$CMTCONFIG/DaVinci.exe ../options/myJob.opts
```

  - C++ like syntax
  - Example

```
Alg1.MassWindow = 10.* GeV;
Alg2.MassWindow = 500.;    // Default is MeV
```

Object name (Instance, not class)

.Property name

= Property value;

Job Options: Conventions

- Many algorithms need many options
  - Options go along with code
    - New code release may need different options
    - Must be configurable with cmt
  - Need for conventions

LHCb conventions

- Job options files of LHCb applications organize sequencing of algorithms

```
ApplicationMgr.DLLs    += { “STAlgorithms" };
ApplicationMgr.TopAlg += {"MCSTDepositCreator/MCITDepCreator"};
```

- 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

```
#include "$STALGORITHMSROOT/options/itDigi.opts“
```

```
MCITDepCreator.tofVector = {25.9, 28.3, 30.5};

ToolSvc.STSignalToNoiseToolIT.conversionToADC = 0.0015;
```

Job Options You Must Know

```
ApplicationMgr.EvtMax        <integer>
ApplicationMgr.DLLs          <Array of string>
ApplicationMgr.TopAlg        <Array of string>
```

- Maximal number of events to execute
- Component libraries to be loaded
- Top level algorithms: “Type/Name”
  “TutorialAlgorithm/Alg1”
  This also defines the execution schedule

Job options printout

Contents of job options files is printed out when Gaudi starts.

- Control printing during processing:

```
#pragma print off // Do not print options defined after this
#pragma print on  // Switch back on
```

- Print a single sorted list of all modified options:

```
#printOptions
```

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?
  - You may want to switch on/off printing at several levels just for one given algorithm, service etc.

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     = 5; // MSG::ERROR
MySvc.OutputLevel          = 4; // MSG::WARNING
MyAlgorithm.OutputLevel    = 3; // MSG::INFO
```

Print everything of INFO level or higher

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;
```

Some units can be used also in job options:

```
SomeAlgorithm.MassWindow = 0.3 * GeV;
```

– List of allowed units in $STDOPTS/units.opts

Exercise

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