ATLAS

Athena

The ATLAS Common Framework

User Guide and Tutorial

Version: 2
Issue: 0
Edition: 2
Status: Draft
ID: 1
Date: 16 August 2001

DRAFT

European Laboratory for Particle Physics
Laboratoire Européen pour la Physique des Particules
CH-1211 Genève 23 - Suisse

Table of Contents

Chapter 1
Introduction

1.1 Purpose of the document

This document is intended as a combination guide and tutorial for users of the Athena control framework. Athena is based upon the GAUDI architecture that was originally developed by LHCb, but which is now a joint development project. This document, together with other information about Athena, is available online at:

http://web1.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture

This version of the Athena User Guide corresponds to Athena release 2.0.0. This is based upon ATLAS GAUDI version 0.7.2, which itself is based upon GAUDI version 7 with some patches.

1.2 Athena and GAUDI

As mentioned above Athena is a control framework that represents a concrete implementation of an underlying architecture. The architecture describes the abstractions or components and how they interact with each other. The architecture underlying Athena is the GAUDI architecture originally developed by LHCb. This architecture has been extended through collaboration with ATLAS, and an experiment neutral or kernel implementation, also called GAUDI, has been created. Athena is then the sum of this kernel framework, together with ATLAS-specific enhancements. The latter include the event data model and event generator framework.

The collaboration between LHCb and ATLAS is in the process of being extended to allow other experiments to also contribute new architectural concepts and concrete implementations to the kernel GAUDI framework. It is expected that implementation developed originally for a particular experiment will be adopted as being generic and will be migrated into the kernel. This has already happened with,

for example, the concepts of auditors, the sequencer and the ROOT histogram and ntuple persistency service.

For the remainder of this document the name Athena is used to refer to the framework and the name GAUDI is used to refer to the architecture upon which this framework is based.

1.2.1 Document organization

The document is organized as follows:

Chapter 2 is the release notes for this version of Athena, detailing changes from previous versions.

Chapter 3 is a short resume of concepts from the architecture.

Chapter 17 discusses physical design issues such as how to access the Gaudi external package from the ATLAS SRT environment, how to deal with component libraries *etc*.

Chapter 6 describes how to access ATLAS data, including Physics TDR data, and Atlfast generated data.

Chapter 7 describes the framework for Monte-Carlo event generators that is available within Athena.

Chapter 7 describes briefly the fast simulation (Atlfast) that uses the output from one of the event generators described in Chapter 6.

Chapter 5 discusses the prototype scripting support which is based upon the Python scripting language.

Chapter 8 discusses the Event Data Model, which is implemented by the StoreGate service.

Chapter 10 describes the proposed data dictionary and related constructs that will be used to auto-generate code for a variety of different purposes, including converters, data browsing tools *etc*.

Chapter 9 summarizes the tutorial examples that are part of the ATLAS software release.

Appendix A contains references and Appendix B is a brief installation guide.

1.3 Conventions

1.3.1 Units

This section is blank for now.

1.3.2 Coding Conventions

This section is blank for now.

1.3.3 Naming Conventions

This section is blank for now.

1.3.4 Conventions of this document

Angle brackets are used in two contexts. To avoid confusion we outline the difference with an example.

The definition of a templated class uses angle brackets. These are required by the C++ syntax, so in the instantiation of a templated class the angle brackets are retained:

```
AlgFactory<UserDefinedAlgorithm> s_factory;
```

This is to be contrasted with the use of angle brackets to denote “replacement” such as in the specification of the string:

```
“<concreteAlgorithmType>/<algorithmName>”
```

which implies that the string should look like:

```
“EmptyAlgorithm/Empty”
```

Hopefully what is intended will be clear from the context.

1.4 Release Notes

Although this document is kept as up to date as possible, Athena users should refer to the release notes that accompany each ATLAS software release for any information that is specific to that release. The release notes are kept in the `offline/Control/ReleaseNotes.txt` file.

1.5 Reporting Problems

Eventually ATLAS will use the Remedy bug reporting system for reporting and tracking of problems. Until this is available, users should report problems to the ATLAS Architecture mailing list at *atlas-sw-architecture@atlas-lb.cern.ch*.

1.6 User Feedback

Feedback on this User Guide, or any other aspects of the documentation for Athena, should also be sent to the ATLAS Architecture mailing list.

Chapter 3
Release notes

3.1 Overview

These release notes identify changes since the previous release, focussing on new functionality, changes that are not backwards compatible, changes in external dependencies, and a brief summary of bugs that have been fixed, or are known to be outstanding.

3.2 New Functionality

3.3 Changes that are not backwards compatible

3.4 Changed dependencies on external software

1. ATLAS release 2.0.0 depends upon ATLAS GAUDI release 0.7.2.

3.5 Bugs Fixed

In general these should be referenced by the appropriate Remedy number, but this is not currently available.

3.6 Known Bugs

None.

Chapter 3
Athena concepts

3.1 Overview

This Chapter summarizes the concepts that are used by the Athena framework, to provide context and a terminology that is used in the rest of the User Guide. Most of the concepts are introduced and described in detail in the GAUDI Architecture Design Document[2], but several other concepts are specific to the ATLAS software environment.

3.2 Athena components

3.2.1 Algorithms

Algorithms form the basic building blocks of user applications, and generally accept input data, manipulate it in some way, and generate new output data. They represent the primary algorithmic part of an application, performing, for example, track finding and fitting, the asssociation of calorimeter hits into clusters and towers, and the association of particle types with tracks and clusters.

Algorithms can be simple or composite, the latter having children that it delegates processing to. These themselves may be composite, allowing quite complicated processing structures to be setup. Algorithms can also act as filters, indicating that a particular event does not meet its selection or filter criteria, and causing downstream Algorithms not to be activated for that event.

3.2.2 Services

Services provide specific capabilities of the framework and as their name implies provide a service to their clients. Histogram and Random Number Generator services are examples. Services hide behind abstract interfaces such that potentially multiple implementations can be provided, the specific implementation being selectable at run time. For example, histograms that have been created by user Algorithms and booked with the Histogram service can be made persistent by one of two Histogram Persistency services. One writes HBOOK files, the other writes ROOT files. The particular implementation can be selected at run time.

3.2.3 Properties

Algorithms and services can have adjustable parameters, called *Properties*, that allow run time configuration. The designer of the algorithm or service will in general decide which of the parameters should be adjustable in this manner. Note that this decision is orthogonal to the design of the C++ class interface, and the designer has flexibility over which adjustable parameters have programmatic adjustability via the public class interface, or run time adjustability by the application user. Properties can be specified via a text file that is read during the startup phase of the application, and, if scripting is enabled, interactively at run time from the scripting language shell.

3.2.4 Job Options files and Python scripts

A job options file is a conventional text file (by default called `jobOptions.txt` in the current directory) that is used to control the configuration of an Athena application at run time. Thus the specification of which Algorithms should be run in which sequence, the particular implementation of services with different possible implementation, and the adjustable properties of framework components can be configured by way of the job options file.

The functionality of job options files is also available using Python scripts. These have the advantage that they can be used both for configuration, and also for interactive sessions.

3.2.5 Data objects and transient stores

Data objects are what are passed between Algorithms, acting as their input and output. In order to reduce the coupling between Algorithms, several so-called transient stores are available that act as the temporary respository for information. Thus an Algorithm will locate input information from a transient store, and write out newly generated information derived from its processing into the transient store, where it can later be retrieved by downstream Algorithms. The different transient stores have different lifetime policies associated with them, particular stores being:

- The event data store or transient event store
- The detector data store

- The histogram store

Retrieving and registering of data with the transient stores is light-weight and does not involve physical copying of data.

3.2.6 Converters

Converters convert objects from one representation to another. One particular use within Athena is to decouple algorithmic code from the underlying persistency mechanism or mechanisms. Thus a set of converters is provided to convert DataObjects within the transient stores to and from an equivalent set of persistent objects for each persistency implementation. An alternative use might also be to convert objects to a graphical representation.

3.2.7 Auditors

Auditors are objects that monitor aspects of other components of the framework. Currently Athena supports auditors for monitoring the following aspects of Algorithms:

- The *NameAuditor* just provides a visible trail of the sequence in which Algorithms are processed for each event, outputting the name of each Algorithm immediately before and after its execution for each event. It is primarily expected to be used as a diagnostic tool.
- The *ChronoAuditor* monitors the cpu usage for every Algorithm, and provides a summary of this at the end of job.
- The *MemoryAuditor* monitors memory usage and provides a warning of possible memory leaks.

3.2.8 Helpers and Tools

Not every algorithmic manipulation need be performed by Algorithm classes which have a particular significance to the framework. Although Algorithms form the basic building blocks that are manipulated and sequenced by the framework in order to process information, it is expected that other helper classes will play an important role in performing the algorithmic processing of data. Thus an Algorithm may locate its desired input data and then delegate further processing to helper classes. Such helper classes will in general not be known to the framework itself, although support for a particular type of helper class, called a Tool, is provided by the framework. Services can also provide helper functions. The Random Number Generator service is an example of this.

The design decision as to whether an Algorithm or a helper should be used for a particular algorithmic operation basically depends upon the granularity of the processing. It would be unreasonable to turn the square root function into a fully fledged Algorithm, just as it would be unreasonable to turn the full reconstruction of an event into a helper. A typical Algorithm will perform a well defined function for a

detector subsystem, such as combining calorimeter hits into clusters or towers. Tools or helper classes are typically used by several Algorithms.

3.3 Packages

The ATLAS software environment is based upon the concept of Packages, being sets of (typically) C++ classes and their interface and implementation files that are grouped together. Each package will, in general, depend on other packages, and will result in the generation of a typically one or more libraries or executables.

Packages are both a management tool and a software configuration tool.

Chapter 4
Establishing a run-time environment

4.1 Overview

This Chapter describes how to establish an environment to allow a user to access Athena-based applications. The details of this will depend upon the particular site, on whether the user is using a CERN computer, and whether AFS is available. Consult your local system administrator for details of how to login and to create a minimal environment. What is described here is the appropriate setup procedures for a CERN user on a CERN machine.

4.2 Establishing a login environment

4.2.1 Commands to establish a bourne-shell or varient login environment

The commands in Listing 4.1 establish a minimal login environment using the bourne shell or varients (sh, bash, zsh, etc.) and should be entered into the .profile or .bash_profile (?) file.

Listing 4.1 Bourne shell and varients commands to establish an ATLAS login environment

```
export ATLAS_ROOT=/afs/cern.ch/atlas
export CVSROOT=:kserver:atlas-sw.cern.ch:/atlascvs
if [ "$PATH" != "" ]; then
    export PATH=${PATH}:$ATLAS_ROOT/software/bin
else
    export PATH=$ATLAS_ROOT/software/bin
fi
source ‘srt setup -s sh‘
```

4.2.2 Commands to establish a c-shell or varient login environment

The commands in Listing 4.2 establish a minimal login environment using the c-shell or varients (csh, tcsh, *etc.*) and should be entered into the .login file.

Listing 4.2 C shell and varients commands to establish an ATLAS login environment

```
setenv ATLAS_ROOT /afs/cern.ch/atlas
setenv CVSROOT :kserver:atlas-sw.cern.ch:/atlascvs
if ( $?PATH ) then
    setenv PATH ${PATH}:$ATLAS_ROOT/software/bin
else
    setenv PATH $ATLAS_ROOT/software/bin
endif
source ‘srt setup -s csh‘
```

4.3 Using SRT to checkout ATLAS software packages

ATLAS software is organized as a set of hierarchical packages, each package corresponding to a logical grouping of (typically) C++ classes. These packages are kept in a centralized code repository, managed by CVS [Ref]. Self-contained snaphots of the package hierarchy are created at frequent intervals, and executables and libraries are created from them. These snapshots are termed *releases*, and in many cases users can execute applications directly from a release of their choice. Each release is identified by a three-component identifier of the form ii.jj.kk (e.g. 1.3.2).

Chapter 6
Scripting

6.1 Overview

Athena scripting support is available in prototype form.The functionality is likely to change rapidly, so users should check with the latest release notes for changes or new functionality that might not be documented here.

6.2 Python scripting service

In keeping with the design philosophy of Athena and the underlying GAUDI architecture, scripting is defined by an abstract scripting service interface, with the possibility of there being several different implementations. A prototype implementation is available based upon the Python[4] scripting language. The Python scripting language will not be described in detail here, but only a brief overview will be presented.

6.3 Python overview

This section is in preparation.

6.4 How to enable Python scripting

Two different mechanisms are available for enabling Python scripting.

1. Replace the job options text file by a Python script that is specified on the command line.
2. Use a job options text file which hands control over to the Python shell once the initial configuration has been established.

6.4.1 Using a Python script for configuration and control

The necessity for using a job options text file for configuration can be avoided by specifying a Python script as a command line argument as shown in Listing 6.1.

Listing 6.1 Using a Python script for job configuration

```
athena MyPythonScript.py                          [1]
```

Notes:

1. The file extension `.py` is used to identify the job options file as a Python script.All other extensions are assumed to be job options text files.

This approach may be used in two modes. The first uses such a script to establish the configuration, but results in the job being left at the Python shell prompt. This supports interactive sessions. The second specifies a complete configuration and control sequence and thus supports a batch style of processing. The particular mode is controlled by the presence or absence of Athena-specific Python commands described in Section 6.8.

6.4.2 Using a job options text file for configuration with a Python interactive shell

Python scripting is enabled when using a job options text file for job configuration by adding the lines shown in Listing 6.2 to the job options file.

Listing 6.2 Job Options text file entries to enable Python scripting

```
ApplicationMgr.DLLs   += { "SIPython" };                          [1]
ApplicationMgr.ExtSvc += { "PythonScriptingSvc/ScriptingSvc" };   [2]
```

Notes:

1. This entry specifies the component library that implements Python scripting. Care should be taken to use the "+=" syntax in order not to overwrite other component libraries that might be specified elsewhere.
2. This entry specifies the Python scripting implementation of the abstract Scripting service. As with the previous line, care should be taken to use the "+=" syntax in order not to override other services that might be specified elsewhere.

Once the initial configuration has been established by the job options text file, control will be handed over to the Python shell.

It is possible to specify a specific job options configuration file at the command line as shown in Listing 6.3.

Listing 6.3 Specifying a job options file for application execution

```
athena [job options file]                              [1]
```

Notes:

1. The job options text file command line argument is optional. The file `jobOptions.txt` is assumed by default.
2. The file extension `.py` is used to identify the job options file as a Python script. All other extensions are assumed to be job options text files. The use of a Python script for configuration and control is described in Section 6.4.1.

6.5 Prototype functionality

The functionality of the prototype is limited to the following capabilities. This list will be added to as new capabilities are added:

1. The ability to read and store basic Properties for framework components (Algorithms, Services, Auditors) and the main ApplicationMgr that controls the application. Basic properties are basic type data members (int, float, *etc*.) or SimpleProperties of the components that are declared as Properties via the declareProperty() function.
2. The ability to retrieve and store individual elements of array properties.
3. The ability to specify a new set of top level Algorithms.
4. The ability to add new services and component libraries and access their capabilities
5. The ability to specify a new set of members or branch members for Sequencer algorithms.
6. The ability to specify a new set of output streams.
7. The ability to specify a new set of "AcceptAlgs", "RequireAlgs", or "VetoAlgs" properties for output streams.

6.6 Property manipulation

An illustration of the use of the scripting language to display and set component properties is shown in Listing 6.4:

Listing 6.4 Property manipulation from the Python interactive shell

```
>>>Algorithm.names                                                    [1][2]
('TopSequence', 'Sequence1', 'Sequence2')

>>> Service.names                                                    [3]
('MessageSvc', 'JobOptionsSvc', 'EventDataSvc', 'EventPersistencySvc',
'DetectorDataSvc', 'DetectorPersistencySvc', 'HistogramDataSvc',
'NTupleSvc', 'IncidentSvc', 'ToolSvc', 'HistogramPersistencySvc',
'ParticlePropertySvc', 'ChronoStatSvc', 'RndmGenSvc', 'AuditorSvc',
'ScriptingSvc', 'RndmGenSvc.Engine')

>>> TopSequence.properties                                           [4]
{'ErrorCount': 0, 'OutputLevel': 0, 'BranchMembers': [],
'AuditExecute': 1, 'AuditInitialize': 0, 'Members':
['Sequencer/Sequence1', 'Sequencer/Sequence2'], 'StopOverride': 1,
'Enable': 1, 'AuditFinalize': 0, 'ErrorMax': 1}

>>> TopSequence.OutputLevel                                          [5]
'OutputLevel': 0

>>> TopSequence.OutputLevel=1                                        [6]

>>> TopSequence.Members=['Sequencer/NewSeq1', 'Sequencer/NewSeq1']   [7]

>>> TopSequence.properties
{'ErrorCount': 0, 'OutputLevel': 1, 'BranchMembers': [],
'AuditExecute': 1, 'AuditInitialize': 0, 'Members':
['Sequencer/NewSeq1', 'Sequencer/NewSeq1'], 'StopOverride': 1,
'Enable': 1, 'AuditFinalize': 0, 'ErrorMax': 1}

>>> theApp.properties                                                [8]
{'JobOptionsType': 'FILE', 'EvtMax': 100, 'DetDbLocation': 'empty',
'Dlls': ['HbookCnv', 'SI_Python'], 'DetDbRootName': 'empty',
'JobOptionsPath': 'jobOptions.txt', 'OutStream': [],
'HistogramPersistency': 'HBOOK', 'EvtSel': 'NONE', 'ExtSvc':
['PythonScriptingSvc/ScriptingSvc'], 'DetStorageType': 0, 'TopAlg':
['Sequencer/TopSequence']}
>>>
```

Notes:

1. The ">>>" is the Python shell prompt.
2. The set of existing Algorithms is given by the `Algorithm.names` command.
3. The set of existing Services is given by the `Service.names` command.

4. The values of the properties for an Algorithm or Service may be displayed using the `<name>.properties` command, where <name> is the name of the desired Algorithm or Service.

5. The value of a single Property may be displayed (or used in a Python expression) using the <name>.<property> syntax, where <name> is the name of the desired Algorithm or Service, and <property> is the name of the desired Property.

6. Single valued properties (*e.g.* `IntegerProperty`) may be set using an assignment statement. Boolean properties use integer values of 0 (or FALSE) and 1 (or TRUE). Strings are enclosed in "'" characters (single-quotes) or """ characters (double-quotes).

7. Multi-valued properties (*e.g.* `StringArrayProperty`) are set using "[...]" as the array delimiters.

8. The `theApp` object corresponds to the ApplicationMgr and may be used to access its properties.

6.7 Synchronization between Python and Athena

It is possible to create new Algorithms or Services as a result of a scripting command. Examples of this are shown in Listing 6.5:

Listing 6.5 Examples of Python commands that create new Algorithms or Services

```
>>> theApp.ExtSvc = [ "ANewService" ]
>>> theApp.TopAlg = [ "TopSequencer/Sequencer" ]
```

If the specified Algorihm or Service already exists then its properties can immediately be accessed. However, in the prototype the properties of newly created objects cannot be accessed until an equivalent Python object is also created. This restriction will be removed in a future release.

This synchronization mechanism for creation of Python Algorithms and Services is illustrated in Listing 6.6:

Listing 6.6 Examples of Python commands that create new Algorithms or Services

```
>>> theApp.ExtSvc = [ "ANewService" ]
>>> ANewService = Service( "ANewService" )                    [1]
>>> theApp.TopAlg = [ "TopSequencer/Sequencer" ]
>>> TopSequencer = Algorithm( "TopSequencer" )                [2]
>>> TopSequencer.properties
```

Notes:

1. This creates a new Python object of type Sequencer, having the same name as the newly created Athena Sequencer.
2. This creates a new Python object of type Algorithm, having the same name as the newly created Athena Algorithm.

The Python commands that might require a subsequent synchronization are shown in Listing 6.7:

Listing 6.7 Examples of Python commands that might create new Algorithms or Services

```
theApp.ExtSvc = [...]
theApp.TopAlg = [...]
Sequencer.Members = [...]
Sequencer.BranchMembers = [...]
OutStream.AcceptAlgs = [...]
OutStream.RequireAlgs = [...]
OutStream.VetoAlgs = [...]
```

6.8 Controlling job execution

This is very limited in the prototype, and will be replaced in a future release by the ability to call functions on the Python objects corresponding to the ApplicationMgr (theApp), Algorithms, and Services.

In the prototype, control is returned from the Python shell to the Athena environment by the command in Listing 6.8:

Listing 6.8 Python command to resume Athena execution

```
>>> theApp.Go                                    [1]
```

Notes:

1. This is a temporary command that will be replaced in a future release by a more flexible ability to access more functions of the ApplicationMgr.

This will cause the currently configured event loop to be executed, after which control will be returned to the Python shell.

Typing Ctrl-D (holding down the Ctrl key while striking the D key) at the Python shell prompt will cause an orderly termination of the job. Althernatively, the command shown in Listing 6.9 will also cause an orderly application termination.

Listing 6.9 Python command to terminate Athena execution

```
>>> theApp.Exit                                  [1]
```


This command, used in conjunction with the theApp.Go command, can be used to execute a Python script in batch rather than interactive mode. This provides equivalent functionality to a job options text file, but using the Python syntax. An example of such a batch Python script is shown in Listing 6.10:

Listing 6.10 Python batch script

```
>>> theApp.TopAlg = [ "HelloWorld" ]
    [other configuration commands]
>>> theApp.Go
>>> theApp.Exit
```

Chapter 6
Accessing ATLAS data

6.1 Overview

This chapter discusses how Athena applications can gain access to ATLAS event data. Available data include those generated for the Physics TDR, both in Objectivity and ZEBRA formats, and data generated by the Atlfast fast Monte-Carlo, available in Objectivity and ROOT formats.

6.2 Accessing Physics TDR data from ZEBRA files

Physics TDR event data stored in ZEBRA files is accessed via the ZebraTDRCnvSvc service and associated ZebraTDRCnv converters.

6.2.1 The ZebraTDRCnvSvc service

The ZebraTDRCnvSvc service is specified as the source of input event data using the following lines in the job options file or Python scripts as shown in Listing 6.1a and Listing 6.1b.

Listing 6.1a JobOptions file fragment to access Physics TDR data in Zebra format

```
#include "Atlas_TDR.UnixStandardJob.txt"
```

Listing 6.1b Python script fragment to access Physics TDR data in Zebra format

```
execfile( "Atlas_TDR.UnixStandardJob.py" )
```

The relevant contents of this included file are shown in Listing 6.2a and Listing 6.2b:

Listing 6.2a Fragment from `Atlas_TDR.UnixStandardJob.txt` job options file

```
ApplicationMgr.Dlls   += { "ZebraTDRCnv" };                                  [1]
ApplicationMgr.ExtSvc += { "ZebraTDRCnvSvc",
                           "ZebraTDREventSelector/EventSelector" };          [2]
EventPersistencySvc.CnvServices = { "ZebraTDRCnvSvc" };                      [3]
```

Listing 6.2b Fragment from `Atlas_TDR.UnixStandardJob.py` Python script

```
theApp.Dlls   = [ "ZebraTDRCnv" ]                                            [1]
theApp.ExtSvc = [ "ZebraTDRCnvSvc",
                  "ZebraTDREventSelector/EventSelector" ]                    [2]
EventPersistencySvc.CnvServices = [ "ZebraTDRCnvSvc" ]                       [3]
```

Notes:

1. This specifies that the `ZebraTDRCnv` component library should be loaded. It is important to use the "+=" syntax in a job options file in order to append the new component library to any that might already have been configured. This is not necessary for a Python script.

2. This adds the relevant services to the list of known services. It is important to use the "+=" syntax in a job options file in order to append the new component library to any that might already have been configured. It is not necessary for a Python script.

3. This specifies the conversion service that is to be used.

Several properties allow these services to be configured. They are illustrated using the fragments in Listing 6.3a and Listing 6.3b:

Listing 6.3a Fragment from `Atlas_TDR.UnixStandardJob.txt` job options file

```
ZebraTDRCnvSvc.InputFile = "slug.car";                [1]
EventSelector.readHits   = false;                     [2]
EventSelector.readDigits = true;
EventSelector.calos      = true;
EventSelector.emBarrel   = true;
EventSelector.emEndcap   = true;
EventSelector.hec        = true;
EventSelector.facl       = true;
EventSelector.tile       = true;
EventSelector.muons      = true;
EventSelector.mdt        = true;
EventSelector.rpc        = true;
EventSelector.tgc        = true;
EventSelector.trt        = true;
EventSelector.clusters   = true;
EventSelector.sct        = true;
EventSelector.pixel      = true;
```


Listing 6.3b Fragment from `Atlas_TDR.UnixStandardJob.py` Python script

```
ZebraTDRCnvSvc.InputFile = "slug.car"                [1]
EventSelector.readHits   = 0                         [2]
EventSelector.readDigits = 1
EventSelector.calos      = 1
EventSelector.emBarrel   = 1
EventSelector.emEndcap   = 1
EventSelector.hec        = 1
EventSelector.facl       = 1
EventSelector.tile       = 1
EventSelector.muons      = 1
EventSelector.mdt        = 1
EventSelector.rpc        = 1
EventSelector.tgc        = 1
EventSelector.trt        = 1
EventSelector.clusters   = 1
EventSelector.sct        = 1
EventSelector.pixel      = 1
```

Notes:

1. The file that is specified by the InputFile property is not the Zebra file containing the Physics TDR events, but the SLUG datacard file.
2. These boolean properties control which detector subsystems are read, and whether the hits or digits (or both) are created.

6.3 Accessing Physics TDR data from Objectivity databases

This Section is incomplete.

6.4 Accessing Atlfast data from Objectivity databases

This Section is incomplete.

6.5 Accessing Atlfast data from ROOT files

6.5.1 Storing Atlfast data in ROOT files

The job options file and Python script fragments that control the output of Atlfast event data to ROOT files shown in Listing 6.4a and Listing 6.4b:

Listing 6.4a Fragment from job options file to store Atlfast data in ROOT files

```
ApplicationMgr.Dlls     += { "DbConverters", "RootDb" };             [1]
ApplicationMgr.ExtSvc   += { "DbEventCnvSvc/RootEvtCnvSvc" };        [2]
ApplicationMgr.OutStream = { "AtlfastRoot" };                        [3]

EventPersistencySvc.CnvServices += { "RootEvtCnvSvc" };              [4]

AtlfastRoot.ItemList = { "/Event#999" };                             [5]
AtlfastRoot.Output   = "DATAFILE='AtlfastData.root' TYP='ROOT'";     [6]

RootEvtCnvSvc.DbType = "ROOT";                                       [7]
```

Listing 6.4b Fragment from Python script to store Atlfast data in ROOT files

```
theApp.Dlls      = [ "DbConverters", "RootDb" ]                      [1]
theApp.ExtSvc    = [ "DbEventCnvSvc/RootEvtCnvSvc" ]                 [2]
theApp.OutStream = [ "AtlfastRoot" ]                                 [3]

EventPersistencySvc.CnvServices = [ "RootEvtCnvSvc" ]                [4]

AtlfastRoot.ItemList = [ "/Event#999" ]                              [5]
AtlfastRoot.Output   = "DATAFILE='AtlfastData.root' TYP='ROOT'"      [6]

RootEvtCnvSvc.DbType = "ROOT"                                        [7]
```

Notes:

1. This specifies that the `DbConverters` and `RootDb` component libraries should be loaded. It is important to use the "+=" syntax in a job options file in order to append the new component library to any that might already have been configured. This is not necessary for a Python script.
2. This adds the relevant services to the list of known services. It is important to use the "+=" syntax in a job options file in order to append the new component library to any that might already have been configured. It is not necessary for a Python script.
3. This specifies the ouput stream that is to be used for this conversion service. Multiple output streams (corresponding to different persistency implementations) can be specified.
4. This specifies the conversion service that is to be used.

5. This indicates that all event data below `/Event` in the transient event store should be written out.
6. This specifies the output file that the events will be written to. It is important that the specified file should not already exist. Appending events to an existing output file is not curently supported.
7. This identifies the particular persistency mechanism.

6.5.2 Reading Atlfast data from ROOT files

The job options file and Python script fragments that control the input of Atlfast event data from ROOT files are shown in Listing 6.5a and Listing 6.5b:

Listing 6.5a Fragment from job options file to input Atlfast data from a ROOT file

```
ApplicationMgr.DLLs     += { "DbConverters", "RootDb" };            [1]
ApplicationMgr.ExtSvc   += { "DbEventCnvSvc/RootEvtCnvSvc",
                             "EventSelector" };                     [2]
EventDataSvc.RootEventCLID = 2101;                                  [3]

EventPersistencySvc.CnvServices += { "RootEvtCnvSvc" };             [4]

EventSelector.Input =
   { "DATAFILE='AtlfastData.root' TYP='ROOT' OPT='READ'" };        [5]

RootEvtCnvSvc.DbType = "ROOT";                                      [6]
```

Listing 6.5b Fragment from Python script to input Atlfast data from a ROOT file

```
theApp.DLLs     = [ "DbConverters", "RootDb" ]                      [1]
theApp.ExtSvc   = [ "DbEventCnvSvc/RootEvtCnvSvc",
                            "EventSelector" ]                       [2]
EventDataSvc.RootEventCLID = 2101                                   [3]

EventPersistencySvc.CnvServices = [ "RootEvtCnvSvc" ]               [4]

EventSelector.Input =
   [ "DATAFILE='AtlfastData.root' TYP='ROOT' OPT='READ'" ]         [5]

RootEvtCnvSvc.DbType = "ROOT"                                       [6]
```

Notes:

1. This specifies that the `DbConverters` and `RootDb` component libraries should be loaded. It is important to use the "+=" syntax in a job options file in order to append the new component library to any that might already have been configured. This is not necessary for a Python script.
2. This adds the relevant services to the list of known services. It is important to use the "+=" syntax in a job options file in order to append the new component library to any that might already have been configured. It is not necessary for a Python script.
3. This specifies the type of object that will be created at the root of the transient event store.
4. This specifies the conversion service that is to be used.
5. This specifies the input file that the events will be read from.
6. This identifies the particular persistency mechanism.

Chapter 7
Monte-Carlo event generators

7.1 Overview

Capabilities of the Athena framework have been used to provide a set of Monte-Carlo event generators that can act as the input to a processing chain of generator, simulation, reconstruction and analysis.

The following generators are available:

- Herwig
- Isajet
- Pythia
- Single particle gun

All of these generators share a common output into the transient event store based on the HepMC package and so can be used interchangeably.

These generators are available in the `offline/Generators/GeneratorModules` package hierarchy.

7.2 Herwig

This Section is missing.

7.3 Isajet

The Isajet event generator is available as an `Algorithm` called `IsajetModule` in the `offline/Generators/GeneratorModules` package. As with the other generators, it inserts the generated events into the transient store in HepMC [3] format. See the GenModules documentation for general information. The note refers only to ISAJET specific material. The External/Isajet package is used to set up the paths to the ISAJET library. External/Stdhep is also needed to get the includes and conversions to HEPEVT.

The relevant fragments of the job options text file and Python script to enable this generator are shown in Listing 7.1a and Listing 7.1b:

Listing 7.1a Job Options text file fragment to enable the Isajet event generator

```
ApplicationMgr.Dlls   += { "GeneratorModules", "McEventSelector" };      [1]
ApplicationMgr.ExtSvc += { "McEventSelector/EventSelector" };            [2]
ApplicationMgr.TopAlg  = { "IsajetModule" }                              [3]
IsajetModule.IsajetCommand = { "isadat decay.dat",                       [4]
                               "isapar pars.dat",
                               "isalis out.lis" };
ApplicationMgr.TopAlg += { ... };                                        [5]
```

Listing 7.1b Python script fragment to enable the Isajet event generator

```
theApp.Dlls   = [ "GeneratorModules", "McEventSelector" ]                [1]
theApp.ExtSvc = [ "McEventSelector/EventSelector" ]                      [2]
theApp.TopAlg = [ "IsajetModule" ]                                       [3]
IsajetModule = Algorithm( "IsajetModule" )
IsajetModule.IsajetCommand = [ "isadat decay.dat",                       [4]
                               "isapar pars.dat",
                               "isalis out.lis" ]
theApp.TopAlg = [ ... ]                                                  [5]
```

Notes:

1. Two component libraries are needed for this example. The `GeneratorModules` library contains all the generators described in this Chapter. The `McEventSelector` library contains the particular Event Selector service that provides an empty event skeleton.
2. The `McEventSelector` service used by the example is not one of the standard services so must be declared by this line. Explicit synchronization is necessary when creating a new Service using Python scripting. This restriction will be removed in a future release.
3. Specifies the `IsajetModule` generator as the first Algorithm to be run. Explicit synchronization is necessary when using Python scripting. This requirement will be removed in a future release.

4. The configuration of the Isajet module is established by three files, which are specified using this command. The defaults are as shown.

   The file identified by `isadat` is the decay table. A valid example is located within this package hierarchy in `offline/Generators/test/share/decay.dat`.

   The file identified by `isapar` is the ISAJET parameters file. Examples can be found in this package hierarchy in `offline/Generators/GeneratorModules/doc`. Refer to the ISAJET manual for details of the content of this file.

   The file identified by isalis is the ISAJET output listing file. It should not exist prior to running the job.

5. Other Algorithms to be run downstream of the generator should be specified here.

7.4 Pythia

This package runs Pythia from within Athena and puts the events into the transient store in HepMC format. See the documentation on GenModule for general information. The note refers only to Pythia specific material. The `External/Pythia` package is used to set up the paths to the Pythia library. `External/Stdhep` is also needed to get the conversions to HEPEVT. This works with pythia6.xxx only.

A WARNING. look carefully at the GNUmakefile.in in `Generators/test`. Note that it is essential to have `-lpythia6` as the library; `-lpythia` corresponds to pythia5 and is not supported.

The relevant fragments of the job options text file and Python script to enable this generator are shown in Listing 7.2a and Listing 7.2b:

Listing 7.2a Job Options text file fragment to enable the Pythia event generator

```
ApplicationMgr.Dlls   += { "GeneratorModules", "McEventSelector" };      [1]
ApplicationMgr.ExtSvc += { "McEventSelector/EventSelector" };            [2]
ApplicationMgr.TopAlg  = { "PythiaModule" }                              [3]
PythiaModule.PythiaCommand = { "isadat decay.dat",                       [4]
                               "isapar pars.dat",
                               "isalis out.lis" };
ApplicationMgr.TopAlg += { ... };                                        [5]
```

Listing 7.2b Python script fragment to enable the Pythia event generator

```
theApp.Dlls   = [ "GeneratorModules", "McEventSelector" ]                [1]
theApp.ExtSvc = [ "McEventSelector/EventSelector" ]                      [2]
theApp.TopAlg = [ "PythiaModule" ]                                       [3]
PythiaModule = Algorithm( "PythiaModule" )
PythiaModule.PythiaCommand = [ ]
theApp.TopAlg = [ ... ]                                                  [5]
```

Notes:

1. Two component libraries are needed for this example. The `GeneratorModules` library contains all the generators described in this Chapter. The `McEventSelector` library contains the particular Event Selector service that provides an empty event skeleton.
2. The `McEventSelector` service used by the example is not one of the standard services so must be declared by this line. Explicit synchronization is necessary when creating a new Service using Python scripting. This restriction will be removed in a future release.
3. Specifies the `PythiaModule` generator as the first Algorithm to be run. Explicit synchronization is necessary when using Python scripting. This requirement will be removed in a future release.
4. The configuration of the Pythia module can be specified using the `PythiaCommand` Property. This has the following syntax:

   ["common_block_name, variable_name, index, value",
   "common_block_name, variable_name, index, value", ...]

   [Need to expand on this]
5. Other Algorithms to be run downstream of the generator should be specified here.

7.5 Single particle gun

The SingleParticleGun event generator generates a single particle for each event in three modes:

- Fixed (1). Particles are generated at a fixed p_T, eta and phi.
- Gaussian (2). Particles are generated in a gaussian distribution in each of p_T, eta and phi.
- Flat (3). Particles are generated in a flat distribution in each of p_T, eta and phi.

These three modes can be mixed such that the generation can be configured, for example, to generate single particles with a fixed p_T, a flat distribution in phi and a gaussian distribution in eta.

The properties that configure the generation of fixed particles are shown in Listing 7.3a and Listing 7.3b:

Listing 7.3a JobOptions file fragment to generate events using the single particle gun

```
ApplicationMgr.DLLs   += { "GeneratorModules", "McEventSelector" };    [1]
ApplicationMgr.ExtSvc += { "McEventSelector/EventSelector" };          [2]
ApplicationMgr.TopAlg  = { "SingleParticleGun" );                      [3]

SingleParticleGun.ModePt  = 1;                                         [4]
SingleParticleGun.ModeEta = 2;
SingleParticleGun.ModePhi = 3;

SingleParticleGun.Pt      = 5.0;                                       [5]
SingleParticleGun.MinPt   = 1.0;
SingleParticleGun.MaxPt   = 100.0;
SingleParticleGun.SigmaPt = 0.1;

SingleParticleGun.Eta      =  0.0;                                     [6]
SingleParticleGun.MinEta   = -4.0;
SingleParticleGun.MaxEta   =  4.0;
SingleParticleGun.SigmaEta = 0.1;

SingleParticleGun.Phi      = 0.0;                                      [7]
SingleParticleGun.MinPhi   = 0.0;
SingleParticleGun.MaxPhi   = 6.28318;
SingleParticleGun.SigmaPhi = 0.1;

SingleParticleGun.PdgCode = 211;                                       [8]
ApplicationMgr.TopAlg += { ... };                                      [9]
```

Listing 7.3b Python script fragment to generate events using the single particle gun

```
theApp.DLLs   = [ "GeneratorModules", "McEventSelector" ]              [1]
theApp.ExtSvc = { "McEventSelector/EventSelector" ]                    [2]
theApp.TopAlg = [ "SingleParticleGun" ]                                [3]
SingleParticleGun = Algorithm( "SingleParticleGun" )

SingleParticleGun.ModePt  = 1                                          [4]
SingleParticleGun.ModeEta = 2
SingleParticleGun.ModePhi = 3

SingleParticleGun.Pt      = 5.0                                        [5]
SingleParticleGun.MinPt   = 1.0
SingleParticleGun.MaxPt   = 100.0
SingleParticleGun.SigmaPt = 0.1

SingleParticleGun.Eta      =  0.0                                      [6]
SingleParticleGun.MinEta   = -4.0
SingleParticleGun.MaxEta   =  4.0

SingleParticleGun.SigmaEta = 0.1

SingleParticleGun.Phi      = 0.0                                       [7]
SingleParticleGun.MinPhi   = 0.0
SingleParticleGun.MaxPhi   = 6.28318
SingleParticleGun.SigmaPhi = 0.1

SingleParticleGun.PdgCode = 211                                        [8]
theApp.TopAlg = [ ... ]                                                [9]
```

Notes:

1. This specifies that the `GeneratorModules` component library should be loaded. It is important to use the "+=" syntax in a job options file in order to append the new component library to any that might already have been configured. This is not necessary for a Python script.
2. This adds the McEventSelector service to the list of services. It is important to use the "+=" syntax in a job options file in order to append the new component library to any that might already have been configured. This is not necessary for a Python script.
3. This establishes the `SingleParticleGun` generator as the first Algorithm for each event. It is important *not* to use the "+=" syntax in a job options file in order ensure that this Algorithm is the first in any sequence of Algorithms..
4. The generation mode for each of p_T, eta and phi can be specified to be 1, 2, or 3, corresponding to Fixed, Gaussian and Flat respectively. The default settings are shown.

5. The fixed value or range of values in p_T are specified by the `Pt`, `MinPt` and `MaxPt` properties. The first (`Pt`) is ignored if the generation mode is flat or gaussian, and the last two (`MinPt` and `MaxPt`) are ignored if the generation mode is fixed. The gaussian width is specified by the `SigmaPt` property, which is ignored unless the generation type is gaussian (2). Units are GeV/c. The default settings are shown.
6. The fixed value or range of values in eta are specified by the `Eta, MinEta` and `MaxEta` properties. The first (`Eta`) is ignored if the generation mode is flat or gaussian, and the last two (`MinEta` and `MaxEta`) are ignored if the generation mode is fixed. The gaussian width is specified by the `SigmaEta` property, which is ignored unless the generation type is gaussian (2). The default settings are shown.
7. The fixed value or range of values in phi are specified by the `Phi`, `MinPhi` and `MaxPhi` properties. The first (`Phi`) is ignored if the generation mode is flat or gaussian, and the last two (`MinPhi` and `MaxPhi`) are ignored if the generation mode is fixed. The gaussian width is specified by the `SigmaPhi` property, which is ignored unless the generation type is gaussian (2). Units are radians. The default settings are shown.
8. The generated particle type is specified by its PDG code.
9. Any further Algorithms should be added at this point. It is important to use the “+=” syntax in a job options file in order to append the new Algorithms to any that might already have been configured. This is not necessary for a Python script.

Chapter 8
Fast simulation

8.1 Overview

This Chapter is in preparation.

Chapter 9
Tutorial examples

9.1 Overview

Several example applications are available to illustrate aspects of the Athena framework. These applications all share a single common executable, job options files being used to establish different configurations for each example. Thus they not only illustrate different capabilities of the Athena framework itself, they also illustrate how to create component libraries. A component library manages Algorithms, Services or Converters. Such components are managed at run-time, and provide an open extensibility of the framework.

These examples are contained in the `AthenaExample` and `AthenaCommon` subpackages of the `Control` package within the ATLAS Software Release environment.

Please check with the Release Notes (`Control/ReleaseNotes.txt`) for the appropriate release for any specific instructions that might differ from those given here.

9.2 Building the tutorial examples

In general it should not be necessary to build the tutorial examples since they should be prebuilt by the ATLAS Software Release build system for each ATLAS Release. However, in case the developer wishes to base their own classes on these examples, the instructions are given here.

It is assumed that the normal ATLAS login environment has been established. In addition, the instructions depend on the ATLAS release that is being used. The following code fragments and instructions use <release> as the release number, and this should be replaced by the actual release (*e.g.* 2.0.2). The ATLAS convention is that all packages in a release are tagged with the same tag, being of

the form `offline-ii-jj-kk`, where ii, jj and kk are components of the release number. Thus `offline-01-03-02` is the CVS tag corresponding to release 2.0.2.

Listing 9.1 Building the Tutorial Examples

```
cd <dir>                                  [1]
srt new src <release>                     [2]
mkdir build                               [3]
cd src
cvs checkout -d Control -r offline-ii-jj-kk offline/Control    [4]
[etc. for other packages]
cd ../build
../src/configure                          [5]
make clean
make install
```

Notes:

1. Select or create a directory within which the examples will be created. Replace `<dir>` in the example listings with the appropriate location.
2. This creates the `src` directory and establishes a local release based upon an ATLAS release. Replace `<release>` with the desired ATLAS release number. Note that some of the setup scripts rely on checked out packages being located in the `src` or `work` directories and will not work without some changes (see later) if this convention is not adhered to.
3. This creates the `build` directory which will be used for the actual build process.
4. This checks out the offline/Control package corresponding to ATLAS release `<release>` into the Control directory. Other desired packages can be checked out in a similar manner. The `-d <dir>` option determines which directory the package will be checked out into. By convention, the package name should be used. The -r <tag> option determines which package version will be checked out. Replace that shown in the example by the desired one. For example, tag `offline-01-03-02` corresponds to ATLAS release 1.3.2, but of course a package tag could also be specified.
5. This and subsequent lines build the tutorial examples, resulting in the necessary libraries and common executable being built.

9.2.1 Running the tutorial examples

There are three different phases that are necessary in order to run the tutorial examples. These are:

- Setting up the files, including necessary scripts and job options files.
- Establishing the necessary run-time environment (search paths etc.)
- Selecting the desired job options file and running the common executable.

These phases are described in the following subsections.

9.2.2 Setting up the files for running the tutorial examples

This only needs to be done once. It consists of copying files from the base release area to a local directory, from which the common application will be executed.

Listing 9.2 Setting up the files for running the tutorial examples

```
cd <dir>
srt new src <release>                                          [1]
mkdir run                                                      [2]
cd run
cp $SRT_DIST/<release>/installed/share/AthenaCommon/*.* .      [3]
cp $SRT_DIST/<release>/installed/share/AthenaExamples/*.* .    [4]
cp $SRT_DIST/<release>/installed/share/StoreGate/*.* .         [5]
sh TDRsetup_links.sh                                           [6]
```

Notes:

1. Replace `<release>` in the above by the actual release (*e.g.* 1.3.2).
2. By convention the tutorial examples are executed from the `run` directory situated alongside the `src` directory.
3. This copies common scripts and job options files to the current directory.
4. This copies the tutorial-specific job options files to the the current directory.
5. This copies the StoreGate example job options file to the current directory.
6. This establishes several local files and symbolic links.

9.2.3 Establishing the run-time environment

This needs to be done each time you login.

Listing 9.3 Establishing the run-time environment

```
cd <dir>/run
source setup.csh     [or source setup.sh]         [1]
```

Notes:

1. Choose the appropriate setup script for your login shell.
2. If the packages are not located in the `src` or `work` directory alongside the current directory, it is necessary to setup the ATLAS_RELEASE environment variable to specify the appropriate local release. This environment variable is not otherwise necessary.

9.2.4 Selecting and running the desired tutorial example

Once the required environment has been established, each example may be run by selecting the appropriate job options file and executing the common application.

Listing 9.4 Selecting and running the desired tutorial example

```
cp <job options file> jobOptions.txt     [1]
athena                                   [2]
     or
athena <job options file>                [3]
```

Notes:

1. The appropriate job options file for each example is given in the following sections.
2. As described earlier, a single common executable is used for all examples.
3. If no command line parameters are given, the default job options file location (jobOptions.txt) is assumed by default. Otherwise, the first parameter following the `athena` executable name is assumed to be the location of the job options file.

9.2.5 The Fortran Algorithm example

This example illustrates wrapping of a FORTRAN algorithm within a C++ class such that it can be used as an Algorithm. This example also illustrates how information from properties of the Algorithm may be passed through to the FORTRAN code. The `FortranAlgJobOptions.txt` or the `FortranAlgJobOptions.py` file establishe the configuration for this example.

Listing 9.5 The `FortranAlgorithm.h` file

```
#include "Gaudi/Algorithm/Algorithm.h"
#include <string>

class FortranAlgorithm : public Algorithm {       [1]
 public:
  FortranAlgorithm (const std::string& name, ISvcLocator* pSvcLocator);
  StatusCode initialize();
  StatusCode execute();
  StatusCode finalize();
 private:
  int         m_lun;                              [2]
  std::string m_fileName;
};
```

Notes:

1. The `FortranAlgorithm` class inherits from the `Algorithm` class, and must provide implementations for the `initialize(), execute()` and `finalize()` functions.
2. Two properties, an `int` and a `std::string` are declared.

Listing 9.6 Fragments from the `FortranAlgorithm.cxx` file

```
#include "FortranAlgorithm.h"

extern "C" {                                                    [1]
  void initialize_(const int& lun, const char*, int);
  void execute_(const int& lun);
  void finalize_(const int& lun);
}

FortranAlgorithm::FortranAlgorithm(const std::string& name,
                                   ISvcLocator* pSvcLocator) :
  Algorithm(name, pSvcLocator), m_lun(16), m_fileName("input.data")
{
  declareProperty("LUN", m_lun);                                [2]
  declareProperty("fileName", m_fileName);
}
StatusCode FortranAlgorithm::initialize(){
  initialize_(m_lun, m_fileName.c_str(), m_fileName.size());  [3]
  return StatusCode::SUCCESS;
}
StatusCode FortranAlgorithm::execute() {                      [4]
  execute_(m_lun);
  return StatusCode::SUCCESS;
}
```

Notes:

1. The FORTRAN functions are declared as external "C" functions.
2. The `FortranAlgorithm` constructor declares the private data members as properties.
3. Both properties are passed through to the FORTRAN `initialize()` function.
4. The `m_lun` property is passed through to the FORTRAN `execute()` function.

9.2.6 The Graphics example

This example has been replaced by the `Graphics/GraphicsFromEvent` package, which is described in a separate document. This Section needs to be updated.

This example illustrates the use of the ATLAS graphics framework to access information from the transient event store. It is based upon an Algorithm that acts as the interface to the ATLAS graphics framework. In a future release of Athena, this framework will be available as a Service, increasing the

flexibility of interaction.The `GraphicsJobOptions.txt` file establishes the configuration for this example.

9.2.7 The HelloWorld example

This example just illustrates the output levels of the Message service, and the use of simple properties. It is the simplest example of an Algorithm. The `HelloWorldJobOptions.txt` or the `HelloWorldJobOptions.py` files establish the configuration for this example.

Listing 9.7a The `HelloWorldJobOptions.txt` file

```
#include "Atlas_Gen.UnixStandardJob.txt"                    [1]

// Load relevant libraries
ApplicationMgr.DLLs += { "AthExHelloWorld" };               [2]

// Top algorithms to be run
ApplicationMgr.TopAlg = { "HelloWorld" };                   [3]

//--------------------------------------------------------------
// Set output level threshold (2=DEBUG, 3=INFO, 4=WARNING, 5=ERROR, 6=FATAL )
//--------------------------------------------------------------
MessageSvc.OutputLevel = 2;                                 [4]

//--------------------------------------------------------------
// Algorithms Private Options
//--------------------------------------------------------------

// For the HelloWorld algorithm                             [5]
HelloWorld.MyInt = 42;
HelloWorld.MyBool = true;
HelloWorld.MyDouble = 3.14159;
HelloWorld.MyStringVec = { "Welcome", "to", "the", "Athena", "Tutorial" };
```


Listing 9.7b The `HelloWorldJobOptions.py` file

```
execfile( "Atlas_Gen.UnixStandardJob.py" )                          [1]

# Load relevant libraries
theApp.DLLs = [ "AthExHelloWorld" ]                                 [2]

# Top algorithms to be run
theApp.TopAlg = [ "HelloWorld" ]                                    [3]
HelloWorld = Algorithm( "HelloWorld" )

#--------------------------------------------------------------
# Set output level threshold (DEBUG, INFO, WARNING, ERROR, FATAL )
#--------------------------------------------------------------
MessageSvc.OutputLevel = DEBUG                                      [4]

#--------------------------------------------------------------
# Algorithms Private Options
#--------------------------------------------------------------

# For the HelloWorld algorithm                                      [5]
HelloWorld.MyInt = 42
HelloWorld.MyBool = TRUE
HelloWorld.MyDouble = 3.14159
HelloWorld.MyStringVec = [ "Welcome", "to", "the", "Athena", "Tutorial" ]
```

Notes:

1. This line specifies the standard ATLAS job options include file for general applications. This is for use by all applications not wishing to use the Physics TDR event data. Even though this example application does not access event information, there must be a source of events, and this sets up such a source, of essentially empty events.
2. This line specifies the appropriate component library for this example.
3. This example just has a single Algorithm that will be executed. Explicit synchronization is necessary when creating a new Algorithm using Python scripting. This restriction will be removed in a future release.
4. Job options text files cannot parse symbolic names and therefore the output level threshold must be specified as an integer. Symbolic names for the message service output levels are available within Python scripts
5. These lines override the default values for the properties declared by the `HelloWorld` Algorithm.

9.2.8 The Histogram and Ntuple example

This example illustrates the use of the Histogram and Ntuple services, and also how to select the persistency mechanism whereby the histograms or ntuples may be stored either as HBOOK files, or

ROOT files. The `HistNtupJobOptions.txt` or `HistNtupJobOptions.py` file establishes the configuration for this example.

Listing 9.8a Fragments from the `HistNtupJobOptions.txt` file

```
#include "Atlas_TDR.UnixStandardJob.txt"                          [1]

//load relevant libraries
ApplicationMgr.DLLs += { "AthExHistNtup" };                       [2]

// This next shared library is necessary if ROOT persistency is selected
////ApplicationMgr.DLLs += { "RootHistCnv" };                     [3]

// Select HBOOK or ROOT persistency (NONE is default)
ApplicationMgr.HistogramPersistency = "HBOOK";                    [4]

ApplicationMgr.TopAlg = { "Hist" };                               [5]
//ApplicationMgr.TopAlg = { "Ntup" };

// Specify the appropriate output file type
HistogramPersistencySvc.OutputFile  = "histo.hbook";              [6]
//HistogramPersistencySvc.OutputFile  = "histo.rt";

NTupleSvc.Output = { "FILE1 DATAFILE='tuple1.hbook' OPT='NEW'" }; [7]
```

Listing 9.8b Fragments from the `HistNtupJobOptions.py` file

```
execfile( "Atlas_TDR.UnixStandardJob.py" )                        [1]

# load relevant libraries
theApp.DLLs = [ "AthExHistNtup" ]                                 [2]

# This next shared library is necessary if ROOT persistency is selected
##theApp.DLLs  = [ "RootHistCnv" ]                                [3]

# Select HBOOK or ROOT persistency (NONE is default)
theApp.HistogramPersistency = "HBOOK"                             [4]

theApp.TopAlg = [ "Hist" ]                                        [5]
Hist = Algorithm( "Hist" )
##theApp.TopAlg = [ "Ntup" ]
## Ntup = Algorithm( "Ntup" )

# Specify the appropriate output file type
HistogramPersistencySvc.OutputFile  = "histo.hbook"               [6]
##HistogramPersistencySvc.OutputFile  = "histo.rt"

NTupleSvc.Output = [ "FILE1 DATAFILE='tuple1.hbook' OPT='NEW'" ]  [7]
```

Notes:

1. This line specifies the standard ATLAS job options include file for applications wishing to use the Physics TDR event data. This sets up a standard input file.
2. This line specifies the appropriate component library for this example.
3. It is not necessary to specify a component library for the HBOOK persistency since this is setup implicitly through the standard job options file.
4. Allowed values are "HBOOK" or "ROOT".
5. Select the desired Algorithm; either that for demonstrating the booking and filling of histograms, or that for demonstrating the booking and filling of ntuples. Explicit synchronization is necessary when creating a new Algorithm using Python scripting. This restriction will be removed in a future release.
6. Uncomment the appropriate line to specify the histogram output file.
7. Modify the filename as appropriate to specify the ntuple output file.

9.2.9 The Liquid Argon Reconstruction example

This example is a snapshot of the Liquid Argon reconstruction, and demonstrates chaining of multiple Algorithms, upstream ones of which write data to the transient event store, downstream ones locating and using this data. The `LArRecJobOptions.txt` file establishes the configuration for this example.

9.2.10 The Pixel reconstruction example

Another example based on the pixel reconstruction.

9.2.11 The Sequencer example

This example demonstrates filtering, branching and prescaling based on the Sequencer class. It also illustrates use of Auditors and the Auditor service to monitor Algorithms. The

`SequencerJobOptions.txt` or the `SequencerJobOptions.py` files establish the configuration for this example.

Listing 9.9a A Fragment of the `SequencerJobOptions.txt` file

```
// Load relevant libraries
ApplicationMgr.DLLs += { "StoreGateExample",                          [1]
                         "AthExHelloWorld" };
ApplicationMgr.DLLs += { "GaudiAlg", "GaudiAug", "StoreGate" };
AuditorSvc.Auditors  = { "NameAuditor", "ChronoAuditor" };            [2]

// Tutorial Sequencer Example
ApplicationMgr.TopAlg = { "Sequencer/TopSequencer" };                 [3]
TopSequencer.StopOverride = true;                                     [4]
TopSequencer.Members  = { "Sequencer/Path1", "Sequencer/Path2" };      [5]
Path1.Members         = { "WriteData", "Prescaler/Prescaler1",        [6]
                          "ReadData/ReadData1", "HelloWorld",
                          "EventCounter/Counter1" };
Path2.Members         = { "WriteData", "Prescaler/Prescaler2",
                          "ReadData/ReadData2", "HelloWorld",
                          "EventCounter/Counter2" };

// Setup the filter algorithms
Prescaler1.percentPass          = 20.0;                               [7]
Prescaler2.percentPass          = 50.0;

// Setup the ReadData Algorithms
ReadData1.DataProducer = "WriteData";                                 [8]
ReadData2.DataProducer = "WriteData";
```


Listing 9.9b A Fragment of the `SequencerJobOptions.py` file

```
# Load relevant libraries
theApp.DLLs = [ "StoreGateExample",                                    [1]
                "AthExHelloWorld" ]
theApp.DLLs = [ "GaudiAlg", "GaudiAug", "StoreGate" ]
AuditorSvc.Auditors  = [ "NameAuditor", "ChronoAuditor" ]              [2]

# Tutorial Sequencer Example
theApp.TopAlg = [ "Sequencer/TopSequencer" ]                           [3]
TopSequencer = Algorithm( "TopSequencer )
TopSequencer.StopOverride = 1                                          [4]
TopSequencer.Members  = [ "Sequencer/Path1", "Sequencer/Path2" ]       [5]
Path1 = Algorithm( "Path1" )
Path2 = Algorithm( "Path2" )
Path1.Members         = [ "WriteData", "Prescaler/Prescaler1",         [6]
                          "ReadData/ReadData1", "HelloWorld",
                          "EventCounter/Counter1" ]
Path2.Members         = [ "WriteData", "Prescaler/Prescaler2",
                          "ReadData/ReadData2", "HelloWorld",
                          "EventCounter/Counter2" ]
WriteData  = Algorithm( "WriteData" )
Prescaler1 = Algorithm( "Prescaler1" )
Prescaler2 = Algorithm( "Prescaler2" )
ReadData1  = Algorithm( "ReadData1" )
ReadData2  = Algorithm( "ReadData2" )
HelloWorld = Algorithm( "HelloWorld" )
Counter1   = Algorithm( "Counter1" )
Counter2   = Algorithm( "Counter2" )

# Setup the filter algorithms
Prescaler1.percentPass          = 20.0                                 [7]
Prescaler2.percentPass          = 50.0

# Setup the ReadData Algorithms
ReadData1.DataProducer = "WriteData"                                   [8]
ReadData2.DataProducer = "WriteData"
```

Notes:

1. This example uses several Athena Component libraries, some from Athena/Gaudi itself, others from other tutorial examples. These are declared here. The `StoreGateExample` library contains the `ReadData` and `WriteData` Algorithms, the `AthExHelloWorld` library contains the `HelloWorld` Algorithm, the `GaudiAlg` library contains the `Sequencer`, `EventCounter` and `Prescaler` Algorithms (which are described in detail

in the GAUDI User Guide Chapter 5), the `GaudiAud` library contains the various Auditors (which are described in detail in the GAUDI User Guide Chapter 11), and the `StoreGate` library contains the `StoreGateSvc` which is used by the `ReadData` and `WriteData` Algorithms.

2. The Auditor service is configured to enable the `NameAuditor`, which just prints the name of each Algorithm as it is executed, and the `ChronoAuditor`, which monitors the cpu time usage of each Algorithm.
3. A single instance of the Sequencer class, called `TopSequencer`, is setup as the application top algorithm. Explicit synchronization is necessary when creating a new Algorithm using Python scripting. This restriction will be removed in a future release.
4. The `TopSequencer` is setup so that it will execute all of it's members, irregardless of whether any of them fails any filters.
5. The `TopSequencer` is setup with two members, `Path1` and `Path2`, both of type Sequencer.
6. The membership of `Path1` and `Path2` are setup. Note that the same instance of `WriteData` is included as a member of both of them. Only the first occurence will be executed for each event. Different instances of the Prescaler class are included in each of `Path1` and `Path2`. These cause filtering of events, such that only a specified fraction of them are accepted and pass downstream. The same instance of the HelloWorld class is included as a member of both paths, downstream of the prescalers. If the first prescaler passes the event, the HelloWorld instance will be executed in the first path. It will not be executed again in the second path, even if the second prescaler passes the event. It will be executed in the second path only if the first prescaler rejects the event, and the second one accepts it. The number of events that are accepted by each path will be recorded by the appropriate EventCounter instance.
7. The fraction of events that are accepted by each prescaler are setup.
8. These lines establish keys which the `ReadData` instances use in conjunction with the object type to locate the information created by the `WriteData` Algorithm in the transient store.

9.2.12 The StoreGate example

This example illustrates writing to and reading back from the transient event store using the StoreGate API discussed in detail in Chapter 8. The `StoreGateJobOptions.txt` or `StoreGateJobOptions.py` files establish the configuration for this example.

Listing 9.10a Fragments from the StoreGateJobOptions.txt file

```
ApplicationMgr.Dlls += { "StoreGateExample", "StoreGate" };    [1]

//top algorithms to be run
ApplicationMgr.TopAlg = { "WriteData", "ReadData" };           [2]

//add StoreGate to the list of external services
ApplicationMgr.ExtSvc += { "StoreGateSvc" };                   [3]

ReadData.DataProducer = "WriteData";                           [4]
```

Listing 9.10b Fragments from the StoreGateJobOptions.py file

```
theApp.DLLs = [ "StoreGateExample", "StoreGate" ]              [1]

# Top algorithms to be run
theApp.TopAlg = [ "WriteData", "ReadData" ]                    [2]
WriteData = Algorithm( "WriteData" )
ReadData  = Algorithm( "ReadData" )

# Add StoreGate to the list of external services
theApp.ExtSvc = [ "StoreGateSvc" ]                             [3]
StoreGateSvc = Service( "StoreGateSvc" )

ReadData.DataProducer = "WriteData"                            [4]
```

Notes:

1. Two component libraries are needed for this example. The `StoreGateExample` library contains the `WriteData` and `ReadData` Algorithms. The `StoreGate` library contains the `StoreGateSvc` service.
2. The `WriteData` Algorithm writes some information to the transient event store. The `ReadData` Algorithm locates and reads it back from the store. Explicit synchronization is necessary when creating a new Algorithm using Python scripting. This restriction will be removed in a future release.
3. The StoreGate service used by the example is not one of the standard services so must be declared by this line. Explicit synchronization is necessary when creating a new Service using Python scripting. This restriction will be removed in a future release.
4. The `DataProducer` property of the `ReadData` Algorithm is used as the key for locating the information produced by the `WriteData` Algorithm.

Appendix A
References

[1] GAUDI User Guide
http://lhcb-comp.web.cern.ch/lhcb-comp/Components/Gaudi_v6/gug.pdf

[2] GAUDI - Architecture Design Report [LHCb 98-064 COMP]

[3] HepMC Reference

[4] Python Reference

[5] StoreGate Design Document

Index