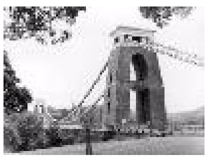
BRUNEL

LHCb Reconstruction Program

User Guide

Version:	1.6
Issue:	1
Edition:	0
Status:	
ID:	[Document ID]
Date:	8 January 2001



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

Document Control Sheet

Table 1 Document Control Sheet

Document	**Title:**	BRUNEL User Guide		
	Version:	1.6		
	Issue:	1		
	Edition:	0		
	ID:	[Document ID]		
	Status:			
	Created:	25 May 2000		
	Date:	8 January 2001		
	Access: :			
	Keywords:			
Tools	**DTP System:**	Adobe FrameMaker	**Version:**	5.5
	Layout Template:	Software Documentation Layout Templates	**Version:**	V1 - 15 January 1999
	Content Template:	--	**Version:**	--
Authorship	**Coordinator:**	M.Cattaneo		
	Written by:	M.Cattaneo		

Document Status Sheet

Table 2 Document Status Sheet

Title:	BRUNEL User Guide		
ID:	[Document ID]		
Version	**Issue**	**Date**	**Reason for change**
1	0	26 May 2000	First draft version
1	1	28 July 2000	Minor changes for Brunel v1r2
1.5	1	17 Nov. 2000	Updated for Brunel v1r5, GaudiSys v6
1.6	1	8 Jan 2001	Updated for Brunel v1r6

Chapter 1
Introduction

1.1 Purpose of this document

This document is a user guide and reference manual for the LHCb reconstruction program, Brunel. It should be useful both to users wishing to run the program, and to programmers wishing to add functionality.

This document does not describe the physics algorithms or the data model.

1.2 What does "Brunel" mean?

All LHCb data processing applications are based on a framework which enforces the GAUDI architecture. Antoni Gaudi [1] was a Catalan architect who greatly influenced the development of Barcelona around the beginning of the nineteenth century. For the reconstruction program we decided to use the name of an engineer. Isambard Kingdom Brunel [2] was a British engineer who greatly contributed to the industrial revolution in the first half of the eighteenth century.

1.3 Editor’s note

This document is a snapshot of the Brunel software at the time of the release of version v1r6. We have made every effort to ensure that the information it contains is correct, but in the event of any discrepancies between this document and information published on the Web, the latter should be regarded as correct, since it is maintained between releases and, in the case of code documentation, it is automatically generated from the code.

We encourage our readers to provide feedback about the structure, contents and correctness of this document and of other Gaudi documentation. Please send your comments to the editor, *Marco.Cattaneo@cern.ch*

Chapter 2
Structure of Brunel

2.1 Brunel Phases

The LHCb reconstruction program, Brunel, is composed of a number of Gaudi algorithms: `BrunelInitialisation`, `BrunelFinalisation` and a number of `BrunelPhase`s.

`BrunelInitialisation` is where all initializations which are independent of BrunelPhase are performed. These can be global program initializations (in the `initialize()` method), or event by event initializations (in the `execute()` method). Note that initializations specific to a given BrunelPhase should not be performed here.

`BrunelFinalisation` is where all finalizations which are independent of BrunelPhase are performed. These can be global program finalizations (in the `finalize()` method), or event by event finalizations (in the `execute()` method). Note that finalizations specific to a given BrunelPhase should not be performed here.

`BrunelPhase` is where the meat of the reconstruction program lies. BrunelPhase is a base class from which actual phases are derived. Each BrunelPhase should be independent of other BrunelPhases: it should be possible to run only one phase, providing of course that event input data in the appropriate format exists[1]. All initializations and finalizations specific to the phase should be performed inside the phase. The following BrunelPhases are currently implemented:

- **`BrunelDigi`** is where simulated RAW Hits are converted into DIGItisings. The output of this phase has the same format as real RAW data coming from the detector[2]. Obviously this phase would not be present when reconstructing real data, and could be moved to the simulation program when reconstructing simulated data. Note that this implies some discipline when designing the DIGItised data model, in particular for what concerns links to Monte Carlo truth information.

1. This is not entirely true in the current version of the reconstruction program, due to the underlying calls to SICBDST routines which do not have this structure.
2. This is not entirely true in the current version of the reconstruction program, due to the underlying use of the SICB event data model, which does not have this structure.

- **`BrunelTrigger`** is where the LHCb trigger decision is applied. The input event data are DIGItisings. The output are also DIGItisings, with the addition of the trigger decision information.
- **`BrunelReco`** is where the first pass reconstruction is carried out. By first pass we mean that the reconstruction algorithms in this phase rely only on DIGItisings and do not require input from the reconstruction of other subdetectors. This restriction can be somewhat relaxed by ensuring that subdetectors are reconstructed in a specific order: those that only require input from the DIGItisings are processed first, those that require input from the reconstruction of other sub-detectors are processed after those sub-detectors.
- **`BrunelFinalFit`** is the second pass reconstruction, to allow for processing which requires input from the reconstruction of several subdetectors.

Note that additional phases could easily be implemented if further reconstruction passes are required.

2.1.1 Instantiating Brunel Phases

Brunel Phases are Gaudi top Algorithms. They are therefore instantiated using the standard Gaudi job option `ApplicationMgr.TopAlg` [3]. Listing 1 shows the value of this option for the current implementation. Note the different phases in lines 2 to 5, which are different instances of the `class BrunelPhase` and will be executed in the order shown

Listing 1 Brunel Top Algorithms as defined in `$BRUNELOPTS/Common.dst1.txt` job options file

```
1:  ApplicationMgr.TopAlg = { "BrunelInitialisation/BrunelInit",
2:                            "BrunelPhase/BrunelDigi",
3:                            "BrunelPhase/BrunelTrigger",
4:                            "BrunelPhase/BrunelReco",
5:                            "BrunelPhase/BrunelFinalFit",
6:                            "BrunelFinalisation/BrunelFinish" };
```

2.2 Brunel sub-detector code

It is expected that sub-detector specific code will be executed inside one or more Brunel Phases. Each Brunel Phase instantiates a Gaudi Sequence for each detector participating in that phase. The instance name of the Sequence follows a specific convention: it is composed of the Phase name (e.g. `BrunelDigi`) followed by the abbreviated sub-detector name (e.g. `MUON`), followed by the string "`Seq`" (e.g. `BrunelDigiMUONSeq`). These Sequences are intended to be the phase specific steering algorithms of the sub-detectors.

Within each Sequence, the sub-detectors are able to instantiate any number of algorithms by simply adding the appropriate job option. For example, to instantiate the `BrunelDigiECAL` and `BrunelDigiHCAL` algorithms in the `BrunelDigiCALOSeq` sequence (and to execute ECAL before HCAL), one would add the following job option::

Listing 2 Example of sequence definition in `$BRUNELOPTS/Common.dst1.txt` job options file

```
BrunelDigiCALOSeq.Members = { "BrunelDigiECAL", "BrunelDigiHCAL" };
```

The advantage of this system is that it is easily extendable and modifiable. To add a new phase, or a new sub-detector, or a new algorithm (or to change any of their names), it is sufficient to make the necessary changes to the job options. No changes are necessary to the Brunel steering code.

The list of sub-detector names and sub-algorithm classes currently implemented is shown in Table 3

Table 3 Sub-detector algorithms currently implemented in Brunel

Sub-detector	Abbreviation	Algorithms implemented

Calorimeters	CALO	`BrunelDigiECAL`
		`BrunelDigiHCAL`
		`BrunelRecoECAL`
		`BrunelRecoHCAL`

Muon Detector	MUON	`BrunelDigiMUON`
Ring Imaging Cherenkov	RICH	`BrunelDigiRICH` `BrunelRecoRICH`

Tracking Detectors	TRAC	`BrunelDigiTRAC`
		`BrunelRecoTRAC`
		`BrunelFinalFitTRAC`

Trigger System	TRIGGER	`BrunelTriggerTRIGGER`
Vertex Locator	VELO	`BrunelDigiVELO`

2.2.1 Instantiating sub-detector sequences

The reason for the naming convention described above is to provide a simple method for selecting which sub-detectors to reconstruct and in which order. It is sufficient to provide a `DetectorList` job option for each phase, containing the list of sub-detectors to be processed in that phase, as shown in Listing 3.

Listing 3 Processing order of sub-detector algorithms in Brunel.

```
BrunelDigi.DetectorList = { "VELO","TRAC","RICH","CALO","MUON" };
BrunelTrigger.DetectorList = { "TRIGGER" };
BrunelReco.DetectorList = { "TRAC" , "RICH" , "CALO" };
BrunelFinalFit.DetectorList = { "TRAC" };
```

2.3 Accessing Gaudi services and data from Brunel

Brunel sub-detector algorithms are instances of Gaudi algorithms. As such they have access to all the services currently implemented in Gaudi, and to all data in the Gaudi data stores. Please refer to the Gaudi user guide [3] for details.

2.4 Adding user code

User code can be added to Brunel in several ways:

1. By providing a Gaudi Algorithm that can be run as a top algorithm outside of a Brunel phase. This would typically be a monitoring algorithm that would analyse the progress of the reconstruction. It can be inserted into the application by declaring it as an additional `ApplicationMgr.TopAlg` in the option shown in Listing 1
2. By providing a new sub-detector algorithm to be called within a given Brunel Phase. It is instantiated by adding the appropriate algorithm name to the Members list of appropriate sequence in the job options file, as shown for example in Listing 2.
3. By replacing an existing sub-detector algorithm.
4. In the current implementation, it is also possible to add a Fortran analysis routine, using the SICB user routines SUINIT, SUANAL, SULAST. SUANAL is called at the end of all event processing.

Chapter 3
Current Implementation

The current version of Brunel implements the "wrapping" of SICBDST Fortran code: Brunel is simply a skeleton within Gaudi which calls the full set of SICBDST Fortran algorithms. There are no C++ algorithms in this version. Version v1r6 of Brunel uses version v235r3 of SICBDST.

3.1 Wrapped SICBDST

The SICBDST code has been wrapped into Brunel Phases and Brunel sub-detector algorithms. Each algorithm calls the corresponding FORTRAN steering routine, as summarised in Table 4

Table 4 List of wrapped SICBDST sub-detector steering routines

Brunel Algorithm	**SICBDST steering routine**
`BrunelInitialisation`	`DETINIT (calls ECINIT, ECDINIT, HCINIT, HCDINIT, MUGINIT, MPINIT)`
`BrunelDigiVELO`	`VSDIGI`
`BrunelDigiTRAC`	`WDDIGI`
`BrunelDigiRICH`	`RIDIGI`
`BrunelDigiECAL`	`ECDIGI`
`BrunelDigiHCAL`	`HCDIGI`
`BrunelDigiMuon`	`MUDIGI`
`BrunelTriggerTRIGGER`	`TRIGGER`
`BrunelRecoTRAC`	`AXTFIT`
`BrunelRecoRICH`	`RIRECO`

Table 4 List of wrapped SICBDST sub-detector steering routines

Brunel Algorithm	**SICBDST steering routine**
`BrunelRecoECAL`	`ECRECO`
`BrunelRecoHCAL`	`HCRECO`
`BrunelFinalFitTRAC`	`AXRECO`
`BrunelFinalisation`	`SUANAL` `RECEVOUT`

Communication between the various FORTRAN algorithms is done, as in SICBDST, via COMMON blocks, in particular the ZEBRA common block. The FORTRAN algorithms are controlled via the SICB data cards file. All data cards recognised by SICBDST are valid, with the exception of cards dealing with input event data (TRIGGERS card, IOPA ’GETX’, ’GETY’, ’GETZ’ cards) and selection of processing steps (SKIP data card). Please refer to the SICB documentation [4] for details

3.2 Choice of execution mode

Four types of excution mode exist for Brunel: with or without Pileup, and using either shared libraries or a statically linked excution. Listing 4 shows the first part of the main Brunel job options file `$BRUNELOPTS/BrunelOptions.txt`. You should uncomment one (and only one!) of the lines 9 to 14, corresponding to the chosen execution mode.

Listing 4 Execution mode specific Brunel options

```
 1:  //--------------------------------------------------------------
 2:  // Setup spillover if wanted This must be the first statement
 3:  //--------------------------------------------------------------
 4:  #include "$BRUNELOPTS/SpillOver.txt"
 5:
 6:  //--------------------------------------------------------------
 7:  // Define the execution mode:
 8:  //--------------------------------------------------------------
 9:  #include "$BRUNELOPTS/Dynamic.dst1.txt"
10:
11:  // Other possibilities are:
12:  // #include "$BRUNELOPTS/Dynamic.dst2.txt"
13:  // #include "$BRUNELOPTS/Static.dst1.txt"
14:  // #include "$BRUNELOPTS/Static.dst2.txt"
```

3.2.1 Choice of pileup mode

Brunel uses the Gaudi implementation of pileup. If you choose a pileup execution mode, you should also select whether you are doing pileup on signal events or on minimum bias events, as shown in Listing 5

Listing 5 Job option for selection of pileup mode

```
// Use LUMISIGNAL if main event is signal, otherwise LUMIMINBIAS
PileUpAlg.PileUpMode = "LUMISIGNAL";
// PileUpAlg.PileUpMode = "LUMIMINBIAS";
```

3.2.2 Inclusion of spillover

It is also possible to read in additional (spillover) events. In order to switch on spillover you should include the file `$BRUNELOPTS/SpillOver.txt` in your job options, as shown in line 4 of Listing 4. This file, reproduced in Listing 6,.shows that, if spillover is included and enabled, only one

Listing 6 Job options for spillover

```
//################################################################
// Options file for adding spillover to Brunel
//================================================================
ApplicationMgr.TopAlg += { "SpillOverAlg" };
SicbEventCnvSvc.enableSpillover = true;
SpillOverAlg.SpillOverMode = "LUMI";
// Enable next two lines to modify number of spillover events.
// Default is Prev = 1, Next = 0; Maximum is Prev=2, Next=2
// SpillOverAlg.SpillOverPrev = 2;
// SpillOverAlg.SpillOverNext = 1;
```

additional event is read in by default, corresponding to the previous beam crossing. Up to a total of 4 spillover events can be read in by changing the values of the `SpillOverAlg.SpillOverPrev` and `SpillOverAlg.SpillOverNext` job options. The spillover events are read into additional branches of the LHCb data model, as shown in Table 5.. Note that the events are merely made available

Table 5

Beam crossing	Event path
Previous	/Event/Prev
One before previous	/Event/PrevPrev
Next	/Event/Next
One after next	/Event/NextNext

in the transient event data store of Gaudi. None of the event information is modified, in particular the time of flight information of the hits is not modified: the labels Prev, Next etc. are for convenience only, it is up to the algorithms using this infomation to add appropriate timing offsets when required.

3.2.2.1 Limitations

The current implementation of spillover has the following limitations:

- It is not possible to add pileup to the spillover events within Brunel. If required, the events read into the spillover input stream should come from a file of previously piled-up events.
- The probability of having a pileup event in a given beam crossing depends on the instantaneous luminosity for that event, which is taken at random from the luminosity distribution of the fill. Currently, independent values of the instantaneous luminosity are taken for each of the pileup events and for the spillover. In a future version the same instantaneous luminosity will be used for all sub-events of a given main event.
- Since spillover events will be combined by digitisers into a single Raw event, only the `/MC` part of the event is provided. Furthermore, only those parts of the `/MC` subevent which have a converter in the `SicbCnv` package are available.

3.3 Input/Output definition

The current version of Brunel uses the Gaudi `SicbEventSelector` to read in event data from a SICBMC RAWH file. The relevant job options are shown in Listing 7

Listing 7 Job Options for event input definition

```
// Input file name (all on one line!)
EventSelector.Input = {"JOBID=’19612’"};
// Number of events to be processed (default is all events)
EventSelector.EvtMax = 100;
// Print event number at each event
EventSelector.PrintFreq = 1;

// Enable next card if you wish to skip some events
// EventSelector.FirstEvent = 3;
```

If you have chosen an execution mode with pileup, you also have to define the file containing the pileup events, as shown in Listing 8

Listing 8 Job Options for pileup

```
// Define the file containing the pileup events
PileUpSelector.JobInput = "JOBID 19065";
```

If you have chosen to enable spillover, you also have to define the file containing the spillover events, as shown in Listing 9. Note that different files must be used for spillover and for pileup.

Listing 9 Job Options for spillover

```
// Define the file containing the pileup events
SpillOverSelector.JobInput = "JOBID 25411";
```

The Gaudi framework does not provide a facility for writing out event data to ZEBRA files. For this reason, Brunel calls the SICB routine RECEVOUT to write out the SICB DST file. The output stream is defined using an IOPA ’SAVX’ data card as for SICBDST [4] (see Listing 10).:

Listing 10 SICB card fto define ZEBRA output file

```
IOPA
     'SAVX' 'XO' '$WORKDIR/Brunel.dst!'
```

In addition, it is possible to write out an object-oriented DST to a ROOT file, using the facilities provided by Gaudi. Please refer to the Gaudi manual [3] for details

3.4 Monitoring

3.4.1 Histograms

In the current version of Brunel, the only predefined histograms are those created by the subdetector code inside SICBDST, control of filling and of output of these histograms is via the SICB data cards in the usual way.

The standard SICBDST checking histograms are also linked into Brunel by default. This is done by line 3 in Listing 11, taken from,the Brunel `requirements` file.

Listing 11 Definition of Brunel application in CMT `requirements` file

```
1:  application Brunel ../Brunel/*.cpp \
2:   $(SICBDSTROOT)/src/sicbvers.F \
3:   $(SICBDSTROOT)/dst/*.F \
4:   ../Brunel/*.F
```

Fill of these histograms is enabled with the following SICB data card:

```
IOPA
    'CHCK' 'HO' '$WORKDIR/Brunel.hbook!'
```

In addition, it is possible for users to define their own histograms inside Gaudi algorithms, using the facilities provided by Gaudi. Such histograms are output by the Gaudi histogram service, to a file defined by the following job option

```
HistogramPersistencySvc.OutputFile  = "histo.hbook";
```

3.4.2 Debug printout

In the current version of Brunel, control of debug printout from the SICBDST Fortran algorithms is via the SICB data cards in the usual way.

In addition, it is possible to define the level of debug printout available from the Gaudi Services and from the Brunel control framework via the standard Gaudi MessageSvc job options:

```
// Global output level
MessageSvc.OutputLevel     = 3;

// Over-ride global level for some algorithms
BrunelInit.OutputLevel     = 2;
BrunelDigiVELO.OutputLevel = 2;
BrunelRecoVELO.OutputLevel = 2;
```

3.4.3 Profiling

Brunel makes use of Gaudi Auditors to monitor the code performance at run time. The following auditors are available:

NameAuditor Prints out the name of an algorithm whenever its `execute()` method is called. Disabled by default.

ChronoAuditor Monitors CPU usage of each algorithm and reports at the end of the job the total and average time per algorithm. Enabled by default.

MemoryAuditor Prints out information on memory usage, in particular whenever the memory allocation changes. Currently only works on Linux. Enabled by default.

The default behaviour of these auditors can be changed using the following job options:

Listing 12 Job options to control default Auditor behaviour

```
//--------------------------------------------------------------
// Enable/Disable some monitoring by setting lines below to true/false
//--------------------------------------------------------------
NameAuditor.Enable   = false;
ChronoAuditor.Enable = true;
MemoryAuditor.Enable = true;

//--------------------------------------------------------------
// Enable/Disable monitoring of methods of individual algorithms
// by setting lines below to true/false
//--------------------------------------------------------------
myAlgorithm.AuditInitialize = false;
myAlgorithm.AuditExceute    = true;
myAlgorithm.AuditFinalize   = false;
```

3.5 Known problems

The following problems and workarounds are known:

- When building Brunel on Linux, you have to `source setup.csh` before typing `gmake`. If you do not do this, `gmake` will not find some of the SICBDST files.

- On NT, the `LHCBHOME` environment variable must contain a path with at least one backslash (e.g. "`V:\cern.ch\lhcb`"). If not, ZEBRA will complain when trying to open the file `$LHCBHOME/sim/data/v111-prob-2d-d0.hbook`. This is a feature of the shift library for Windows..
- Auditors are not currently working in the static exceutable of Brunel.

Chapter 4
Running Brunel

Brunel is implemented as a CMT [5] package, with the following subdirectory structure:

—	`Brunel`	C++ and Fortran source code
—	`doc`	release notes
—	`job`	example job
—	`options`	structure of example steering data cards
—	`mgr`	CMT `requirements` file
—	`Visual`	Visual Studio Workspace

The `job` subdirectory contains an example job for running Brunel on Linux. The options subdirectory contains and example structure of Gaudi job options files (the top file of the structure is `BrunelOptions.txt`) and a SICB data file (`Brunel.cards`). You should customise these files according to your needs.

The files `BrunelOptions.txt` and `Brunel.cards` are picked up by default when you run one of the example jobs on Linux, or inside Visual Studio on NT. To pick up different files, you should modify the following two lines in the `requirements` file

```
# Set the paths for Brunel and SICBDST data cards.
set JOBOPTPATH ${BRUNEOPTS}/BrunelOptions.txt
set SICBCARDS ${BRUNELOPTS}/Brunel.cards
```

Appendix A
References

1 See for example *http://www.gaudiclub.com/ingles/i_vida/i_menu.html* for more information about Antoni Gaudi

2 See for example *http://www.spartacus.schoolnet.co.uk/RAbrunel.htm* for more information about Isambard Kingdom Brunel

3 The GAUDI users guide is available at: *http://cern.ch/lhcb-comp/Components/Gaudi_v6/gug.pdf*

4 The SICB documentation is available at: *http://cern.ch/lhcb-comp/SICB/*

5 CMT documentation is available at *http://cern.ch/lhcb-comp/Support/html/cmt.htm*