# MicroDST

Juan Palacios (Nikhef)

LHCb software week, 15-19 June, 2009

16 June 2009

## Foreword

- I will try to introduce motivate the concept of the MicroDST
- I will omit many technical details for simplicity. For gory details, see presentation of 18/09/2008
  – http://indico.cern.ch/materialDisplay.py?contribId=23&sessionId=4&materialId=slides&confId=25002
  – And the recently decimated Wiki page: https://twiki.cern.ch/twiki/bin/view/LHCb/MicroDST
- The emphasis here is on how to make and read them back

## Contents

- Introduction
- Basic idea
- Components
- Examples
  - Writing the hard way
  - Writing the easy way
  - Reading
- Final remarks

## Introduction

- What is the MicroDST?
  - Idea and first implementation by Ulrich Kerzel
    - A DST with only the information we need in order to do a particular analysis
  - A DST with some information taken out
  - Plus some post-reconstruction physics information added in
  - A framework to allow storing event model objects in memory into a small DST

# Introduction

- Two-fold data reduction (physics analysis context)
  - In a physics analysis, store only relevant information ***but*** only for events that we care about (typically those passing some selecion)
- Format recognised by our LHCb software tools
  - Store this information such that it can be analysed with DaVinci C++ algorithms, GaudiPython, etc.
  - Can act on stored data using standard LHCb physics tools
- Use Gaudi technology to get back to information on original DSTs if necessary
  - Meaning catalogues and SmartRefs

## MicroDST: basic idea

- Store event model objects *of interest* in a magic TES location: refer to this as ***cloning***
  - '/Event/Rec/Vertex/V0' -> '/Event/AbraCadabra/Rec/Vertex/V0'
  - These ***must be*** Event model classes
- Write everything in the magic TES location to a file
  - Store everything under '/Event/AbraCadabra'
- Analyse the file using standard LHCb software
  - Actually, can use any technology you would use on a DST, python, C++, even CINT if you are completely crazy
  - Just remember that not all the info is there

# MicroDST components

- MicroDSTAlgorithms and MicroDSTTools
- User configures MicroDSTAlgorithms
  - MicroDSTAlgorithms may rely on MicroDSTTools to perform ***cloning***
  - MicroDSTTools may own other MicroDSTTools
    - For cases where one Event Model class owns pointers to objects of another Event Model class
    - Ex. Particles and Vertices, RecVertices and Tracks
- Tools should have minimal configuration
- In real life, we have base classes, templates, interfaces… but that only matters if you want to be a MicroDST developer

# MicroDST components

- The user can determine what to write to the MicroDST through a choice MicroDSTAlgorithms and MicroDSTTool implementations
  - More scalable than having one big algorithm and many configuration parameters
- There are some 15 event model classes currently supported
  - These can be written to the MicroDST on one or more ways
    - For example, you can just copy ProtoParticles directly, or you can copy the ProtoParticles pointed at by copied Particles

# MicroDST components

- The magic location prefix is a property of the tools and algorithms
  - MyMDSTAlg.OutputPrefix = “AbraCadabra”
  - Results in objects being cloned from
  - /Event/SomeLocation to /Event/AbraCadabra/SomeLocation
  - Furthermore, related quantities also get copied with the same OutputPrefix
  - Example:
    - Cloning a particle from /Event/Phys/B0Cand/Particles to /Event/Phys/microDST/B0Cand/Particles will result in all the daughters and vertices being copied into similar TES locations. /Event/Phys/DaughtersA/Particles to /Event/microDST/DaughtersA/Particles, etc.
- For standard use, **OutputPrefix** can be left to its default, “microDST”

## Examples

Writing a “typical” MicroDST for physics analysis

# Typical MicroDST contents

- Primary vertices
  - Plus relations to B candidate, if existing
- RecHeader
- ODIN
- Selected B candidates
  - Decay products
  - Decay vertices
  - Origin vertex
  - ProtoParticle
- Re-fitted PVs and relations to candidate B
- Flavour tag
- MC particles associated to all B candidates + decay products
  - Relevant MCVertices
  - Particle -> MCParticle associations

## Write a MicroDST: set-up

A few things need setting up first

```
importOptions(“$STDOPTS/LHCbApplication.py”)
importOptions(“SomeSelection.py”)
mainLocation = “Phys/SelectionLocation” #need to know this
importOptions('$MICRODSTOPTS/MicroDSTStream.py')
MicroDSTStream=OutputStream('MicroDSTStream')
ApplicationMgr().OutStream.append(MicroDSTStream)
MicroDSTStream.Output = “MyFirstMicroDST.dst”
MySelection = GaudiSequencer(“MainSequenceName”) # need to know seq name
MicroDSTStream.AcceptAlgs.append( MySelection.name() )
```

This will write the MicroDST stream to a file whenever the selection sequence passes an event

## Write a MicroDST: simple objects

We start with some simple objects, that is, Event Model types that don’t have any pointers to other event model objects on the TES

```
from Configurables import CopyRecHeader, CopyODIN, CopyHltDecReports
MySelection.Members += [CopyRecHeader()]
MySelection.Members += [CopyODIN()]
MySelection.Members += [CopyHltDecReports()]
```

## Write a MicroDST: Particles

Now we copy some more complicated objects. Here, the MicroDSTAlgorithm needs some configuration

```
from Configurables import CopyParticles
copyParticles = CopyParticles('CopyParticles')
copyParticles.InputLocation = mainLocation+"/Particles”
# uncomment the following if you want ProtoParticle cloning
# The vertex cloning is actually on by default
#copyParticles.addTool(ParticleCloner, name='ClonerType')
#copyParticles.ParticleCloner.addTool(VertexCloner,
   name='IClonerVertex')
#copyParticles.ParticleCloner.addTool(ProtoParticleCloner,
   name='ICloneProtoParticle’)
MySelection.Members += [copyParticles]
```

This results in a clone of the particles, their origin vertices, decay vertices, decay products

# Write a MicroDST: PVs

Now we copy some more complicated objects. Here, the MicroDSTAlgorithm needs some configuration

```
from Configurables import CopyPrimaryVertices,
   CopyParticle2PVRelations
# Copy all primary vertices
copyPV=CopyPrimaryVertices('CopyPrimaryVertices')
copyPV.OutputLevel = 4
MySelection.Members += [copyPV]
# copy PV->Particle link
# This will only copy related PVs, and only if they haven't been
# copied before
copyP2PVRel = CopyParticle2PVRelations()
copyP2PVRel.InputLocation = mainLocation+"/Particle2VertexRelations"
copyP2PVRel.OutputLevel=4
MySelection.Members += [copyP2PVRel]
```

## Write a MicroDST: Fancier stuff (1)

Now we run an algo to get FlavourTags and put them on the MicroDST

```
# first, make the flavour tags
importOptions('$FLAVOURTAGGINGOPTS/BTaggingTool.py')
BTagAlgo = BTagging('BTagging')
BTagAlgo.InputLocations=[mainLocation]
BTagLocation = mainLocation+"/Tagging"
BTagAlgo.TagOutputLocation = BTagLocation
MySelection.Members += [BTagAlgo]
# Now copy them to the MicroDST
from Configurables import CopyFlavourTag
copyFlavTag = CopyFlavourTag()
copyFlavTag.InputLocation = BTagLocation
MySelection.Members += [copyFlavTag]
```

## Write a MicroDST: Fancier stuff (2)

Make Particle->MCParticle associations

and put them on the MicroDST

```
# first, make the Particle->MCParticle relations table
# This will make a relations table in mainLocation+"/P2MCPRelations"
p2mcRelator = P2MCRelatorAlg()
p2mcRelator.ParticleLocation = mainLocation+'/Particles'
MySelection.Members += [p2mcRelator]
# Now copy relations table + matched MCParticles to MicroDST
copyP2MCRel = CopyParticle2MCRelations()
copyP2MCRel.addTool(MCParticleCloner)
copyP2MCRel.MCParticleCloner.addTool(MCVertexCloner,
                                     name = 'ICloneMCVertex')
copyP2MCRel.MCParticleCloner.OutputLevel=4
copyP2MCRel.InputLocation = mainLocation+"/P2MCPRelations"
MySelection.Members += [copyP2MCRel]
```

## Write a MicroDST: even fancier stuff

Re-fit PVs, sort relations according to IRelatedPVFinder weighting logic (IP chi2) and store relations and re-fitted PVs on MicroDST

```
# first, re-fit the PVs and make relations table
PVReFitter = PVReFitterAlg("PVReFitterAlg")
PVReFitter.ParticleInputLocation = mainLocation+"/Particles”
PVReFitter.VertexOutputLocation = mainLocation+"/RefittedVertices”
p2ReFitPVRelationsLoc = mainLocation+"/Particle2ReFittedVertexRelations”
PVReFitter.P2VRelationsOutputLocation = p2ReFitPVRelationsLoc
MySelection.Members += [PVReFitter]
# now Take Particle->PV relations table from previous step and produce
# new table, sorted according to an IRelatedPVFinder
PVRelator = PVRelatorAlg()
PVRelator.P2PVRelationsInputLocation = p2ReFitPVRelationsLoc
p2pvSortedRelationsLoc = mainLocation + "/P2ReFitPVSortedRelations”
PVRelator.P2PVRelationsOutputLocation = p2pvSortedRelationsLoc
MySelection.Members += [PVRelator]
```

## Write a MicroDST: even fancier stuff (2)

```
# Finally, copy sorted table. This also copies the re-fitted PVs.
copyP2RefitPVRel = CopyParticle2PVRelations("CopyP2RefitPVRelations")
copyP2RefitPVRel.InputLocation = p2pvSortedRelationsLoc
copyP2RefitPVRel.OutputLevel=4
MySelection.Members += [copyP2RefitPVRel]
```

- Note that only this last step is using MicroDST code. The rest is all (new) standard DaVinci. As with the FlavourTags and the related MCParticle cloning, we make a clear distinction between the code that puts stuff on the standard TES and code that clones it into the MicroDST
- All the examples come from **`Ex/MicroDSTExample/options/TestMicroDSTMake.py`**, which is a good place to look at how the algorithms can be set up and configured. Code snippets also appear in the doxygen documentation of almost all of the MicroDSTAlgorithms.

## Write a MicroDST the easy way

- If you just want to store standard stuff and don't care about changing the default configuration of MicroDSTTools, you can use the new **`PhysMicroDST()`** configurable:

```
from Configurables import DaVinci, PhysMicroDST
importOptions("$STDOPTS/PreloadUnits.opts")
importOptions( ″SomeSelection.py")
MySelection = GaudiSequencer("SequencerName)
conf = PhysMicroDST()
conf.OutputPrefix = "MicroDST"
conf.MicroDSTFile = "MyTestMDST_MC_newConf.dst"
conf.MicroDSTSelectionAlg = ″MainAlgoName"
conf.CopyL0DUReport = False
conf.CopyHltDecReports = False
conf.CopyMCTruth = True
conf.CopyBTags = True
dv = DaVinci()
dv.EvtMax = 500
dv.UserAlgorithms = [MySelection]
dv.Input =  [ some list of DST files ]
```

- See **`Ex/MicroDSTExample/options/TestMicroDSTMakeNewConf.py`**

## Reading the MicroDST

- Since the MicroDST is a DST, all the standard LHCb ways or reading DSTs can be used. Personally, I do everything in GaudiPython, but it is also possible to access it directly in C++

# Reading the MicroDST in C++

- Since everything is stored under '/Event/microDST' by default, it is enough to use the RootInTES property of algorithms:

```
DaVinciMainSeq.RootOnTES  = "/Event/microDST/"
YourSeq.RootOnTES  = "/Event/microDST/"
TestSeq.Members += ["YourAlg"]
YourAlg.PhysDesktop.InputPrimaryVertices = "microDST/Rec/Vertex/Primary"
YourAlg.PhysDesktop.OnOfflineTool.OfflinePVLocation  = "/Event/microDST/Rec/
   Vertex/Primary"
```

- However, one must always remember that some tools or algorithms access information that isn't on the MicroDST
- If access is really needed, a POOL/ROOT catalogue must be provided to navigate to the original files that were used to make the MicroDST. But this negates some of the advantages of the MicroDST and so is discouraged
- I'm not an expect in C++ reading so I'll stop here

## Reading the MicroDST in GaudiPython

- The best way to illustrate how to do this is by example
- See `Ex/MicroDSTExample/scripts/MicroDSTReadingExample.py`
- A series of helper functions and functors are provided in the MicroDSTExample module
  - Including HistoUtils, borrowed from a future release of GaudiPython
  - These are generic, not limited to MicroDST reading, so I'll probably put them in a DaVinciPython module at some point
- It is necessary to know some key TES locations, like those of the candidate Particles, and necessary relations tables
- Other than that, it is pretty straight forward

# Reading in GaudiPython (2)

## Set up some TES locations and configuration

```
locationRoot = '/Event/microDST'
selection = 'DC06selBs2JpsiPhi_unbiased'
microDSTFile = ['MyFirstMicroDST.dst']
# set up some useful paths of locations on the MicroDST
selectionPath = locationRoot +  '/Phys/' + selection
particlePath = selectionPath + '/Particles'
particle2mcPath = selectionPath + '/P2MCPRelations'
stdVertexAssocPath = selectionPath + '/Particle2VertexRelations'
refitVertexAssocPath = selectionPath + '/P2ReFitPVSortedRelations'
mcParticlePath = locationRoot+'/MC/Particles'
flavTagPath = selectionPath + "/Tagging"
pvLocation = locationRoot+"/Rec/Vertex/Primary"
# we will use one of the new MC association tools with a non-default
# configuration, so we need to get it's configurable
myP2MCP = MCMatchObjP2MCRelator()
myP2MCP.OutputLevel=4
myP2MCP.MCParticleDefaultLocation = mcParticlePath
myP2MCP.RelTableLocations = [particle2mcPath]
```

## Reading in GaudiPython (3)

Initialise the application and get some services

```
From MicroDSTExample import Helpers, Functors, HistoUtils
lhcbApp = LHCbApp()
lhcbApp.DDDBtag = 'default'
lhcbApp.CondDBtag = 'default'

appMgr = AppMgr(outputlevel=4)
appMgr.config( files = ['$STDOPTS/LHCbApplication.opts',
                        '$GAUDIPOOLDBROOT/options/GaudiPoolDbRoot.opts',
                        '$DDDBROOT/options/DC06.opts'])
appMgr.initialize()
appMgr.ExtSvc += ['LHCb::ParticlePropertySvc']
appMgr.HistogramPersistency = "ROOT"

evtSvc = appMgr.evtSvc()
toolSvc = appMgr.toolsvc()
evtSel = appMgr.evtSel()
nextEvent = Helpers.NextEvent(appMgr) # helps with event looping
pp = Helpers.PartPropSvc(appMgr)
ppSvc = pp
```

## Reading in GaudiPython (4)

Initialise the application and get some services. Book some plots

```
# get an instance of PropertimeFitter
properTimeFitter = toolSvc.create('PropertimeFitter',
                                  interface='ILifetimeFitter')
# get an instance of MCMatchObjP2MCRelator
MCAssocTool = toolSvc.create('MCMatchObjP2MCRelator',
                             interface='IP2MCP')
# open the MicroDST
evtSel.open(microDSTFile)
# book plots for a Bs for example. Normally I'd look over PIDs of more particles
histoPath = "MicroDST/Histos/"
pp=ppSvc(531)
particleName = pp.particle()
pdgMass = pp.mass()
massPlot = HistoUtils.book ( histoPath+"Particle/M_"+particleName,
                             particleName + " mass",
                             100, pdgMass-100., pdgMass+100,)
massResPlots = HistoUtils.book( histoPath+"Particle/MResol_"+particleName,
                               particleName + " mass resolution",
                               100, -100., 100,)
```

## Reading in GaudiPython (5)

### Book more plots

```
# proper time plots
propTimePlot = HicroUtils.book(histoPath+"Particle/PropRes”,"Proper time",
                               100, -2., 5.)
refitPropTimePlot = HistoUtils.book(histoPath+"Particle/RefitPropTime",
                                    "Re-fitted PV proper time”, 100, -2., 5.)

propTimeResPlot = HistoUtils.book(histoPath+"Particle/PropTimeRes",
                                  "Proper time resolution",
                                  100, -0.2,0.5)
refitPropTimeResPlot = book(histoPath+"Particle/RefitPropTimeRes",
                            "Re-fitted PV proper time resolution",
                            100, -0.2,0.5)

nPVPlot = HistoUtils.book( histoPath+"PV/nPV”, "# of primary vertices",
                           5, -0.5, 4.5 )
```

## Reading in GaudiPython (6)

Loop and fill histograms (some safety checks omitted for clarity)

```
# proper time plots
while ( nextEvent() ) :
 PVs = evtSvc[pvLocation]
 nPrimaryVertices += PVs.size()
 HistoUtils.fill( nPVPlot, PVs.size() )
 particles = evtSvc[particlePath]
# get P-> refitted PV relations and make a “best” PV functor
 refitBestVertexAssoc = evtSvc[refitVertexAssocPath]
 refitBestVertexFun = Functors.BestVertex(refitBestVertexAssoc)
# make a proper time functor using the refitted PVs
 refitTauFunc = Functors.PropTime(refitBestVertexFun, properTimeFitter)
# wrap the MCAssociator into a functor
 MCAssocFun = Functors.MCAssociator(MCAssocTool, verbose=False)
# loop over particles and plot proper time resolution using re-fitted PVs
 for p in particles:
   refitPropTime = refitTauFunc(p)
   HistoUtils.fill(refitPropTimePlot, refitPropTime)
   assocMCPart = MCAssocFun(p)
   stdPropTimeRes = stdPropTime-Helpers.properTimeMC(assocMCPart)
   HistoUtils.fill(propTimeResPlot, stdPropTimeRes)
```

# Reading in GaudiPython (7)

- In GaudiPython we have Gaudi tools, and services
- In the MicroDST we have Particles, vertices, other reconstructed objects
- Can write bits of python to do extra number crunching
- Can really do reasonable analysis without requiring C++
- Can use other software available through python
  - RooFit for example.
- The main difference between this and a "standard" GaudiPython job running on a DST is that we have the reconstructed Particles, and have access to extra information in the form of relations tables
  - See Particle -> MC and Particle -> PV matching

## And finally…

- I have left out many technical details
- Some complexities in the code are there so new event model classes can be added easily to the MicroDST
  - Can be as easy as a typedef
- The MicroDST wiki has been updated recently
  - https://twiki.cern.ch/twiki/bin/view/LHCb/MicroDST
  - It includes explanations about everything, examples, list of supported Event Model classes
  - Last changes seem to be lost due to server failure and I am pretty pissed off about it

## Extra slides

# Data reduction

- For this typical MicroDST
  - 13KB/event
- Full DST
  - 500KB/event
- So per event, 2.6% of DST size
- Add data reduction factor
  - For this J/Psi K* signal selection, factor 10
- ~0.26% wrt original DSTs
  - Clearly this depends on the selection and the input data sample!