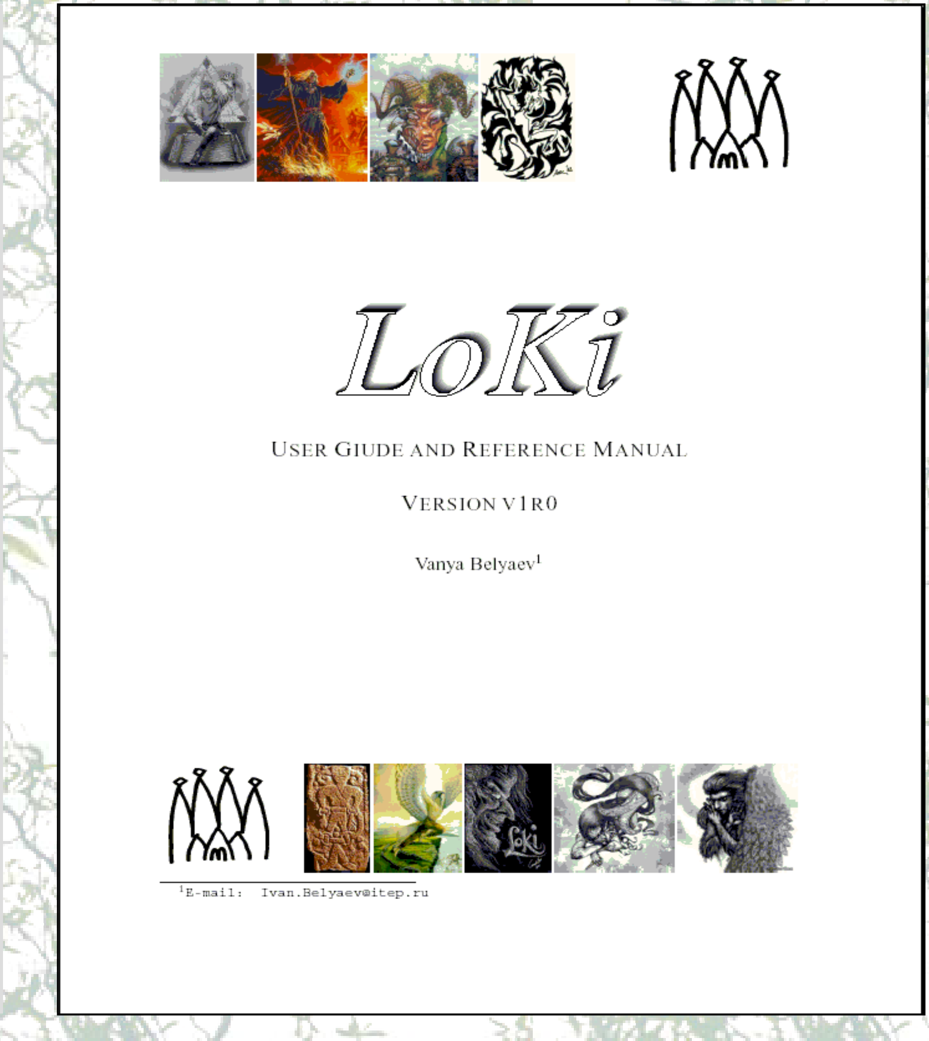
LoKi's Cook-book: Writing analysis algorithms in C++

Vanya Belyaev
LAPP/Annecy & ITEP/Moscow

Outline

- LoKi
  - v3r5
- Current functionality & recipies
- Future steps
- Summary



LoKi

USER GIUDE AND REFERENCE MANUAL

VERSION V1R0

Vanya Belyaev[1]

[1] E-mail: Ivan.Belyaev@itep.ru

LoKi

C++ *Toolkit for user friendly Physics Analysis*

- Available for users from begin of 2003
  - The first analysis has been reported March 2003
    - Benoit Viaud: $B^0 \rightarrow \phi K_S$
- Used for few **TDR** studies in 2003
- In use for some **DC04** selections/stripping
- In use for private studies
- Mailing list: **lhcb-loki@cern.ch**
- See detailed presentations:
  - Software week: June 4th 2k+4
  - LHCb-light: June 3rd 2k+3

LoKi

The major design criteria

- Locality
  - Introduce and use objects in local scope
  - One file
  - One method
  - One screen
- Compact code
- Safety
  - No need in `new, delete`
- "Standard"
  - Use STL idioms & semantics

- The details can be found in "LoKi User Guide & Reference Manual"

```
getpack Doc/LoKiDoc head
cd Doc/LoKiDoc/v<X>/cmt
source setup.[c]sh
cd ../doc
make
```

+DoxyGen documentation

LoKi

- To be discusses today:
  - LoKi & DaVinci
  - LoKi basic
  - MC matching
  - Loops & Charge-blind loops
  - Recipies on every day
- Out of today's discussion
  - Customization of LoKi
  - Future steps

    "to-do" list from June 6th is still full

LoKi & DaVinci

- LoKi is a toolkit for DaVinci
  - Code : LoKi
  - Job Configuration & steering: DaVinci
- All user code is placed in the body of algorithm, which inherits from `LoKi::Algo`, which inherits from `GaudiTupleAlg/GaudiHistoAlg/GaudiAlgorithm` chain
- Only one mandatory method `analyse()` needs to be redefined
  - majority of mandatory and tedious stuff is hidden by preprocessor MACROs

“Hello, World”

```
#include “LoKi/LoKi.h”

LOKI_ALGORIHTM( MyAlg )
{

 info() << “Hello, World” << endreq ;

 return StatusCode::SUCCESS ;
};
```

- Algorithm body,
- implementation of constructor & destructor,
- factories
- `MyAlg::analyse()`

6 lines,

1 functional line

From (to?) base classes:

- Generic access to data, tools and services

```
 get<TYPE>    (…)
 tools<TYPE> (…)
 svc<TYPE>    (…)
```

- Printout & error counts:

```
info(), debug() , error() , fatal(), …
Error(…), Warning(…)
```

- Histograms, NTuples and Event Collections

```
plot(…)
nTuple()
evtCol()
```

DaVinci tools

- Almost all DaVinci tools are available directly with compatible methods:

```
IMassVertexFitter*   massVertexFiter   ( const size_t index = 0 ) const ;
IVertexFitter*       vertexFitter      ( const size_t index = 0 ) const ;
IDirectionFitter     directionFitter   ( const size_t index = 0 ) const ;
ILifetimeFiter*      lifetimeFitter    ( const size_t index = 0 ) const ;
IParticleStuffer*    particleStuffer   ( const size_t index = 0 ) const ;
IParticleFilter*     particleFilter    ( const size_t index = 0 ) const ;
IFilterCriterion*    filterCriterion   ( const size_t index = 0 ) const ;
IPhysDesktop*        desktop              () const ;
IGeomDispCalculator* geomDispCalculator   () const ;
IDecayFinder*        decayFinder          () const ;
IMcDecayFinder*      mcDecayFinder        () const ;
IPhotonTool*         photonTool           () const ;
```

Basic types

- 4 types of basic "**objects**":

```
    Particle, Vertex, MCParticle, MCVertex
```

- "**Function**" : functor which gets as argument the pointer to the "**object**" and returns double

```
    Func, VFunc, MCFunc, MCVFunc     (interface)
    Fun , VFun , MCFun , MCVFun      (assignable)
```

- "**Cut/Predicate**": functor, which gets as an argument the pointer to the "**objects**" and returns bool

```
    Cuts, VCuts, MCCuts, MCVCuts     (interface)
    Cut , VCut , MCCut , MCVCut        (assignable)
```

- "**Range**": a lightweight representation (STL compliant) of container/sequence of "**objects**"

```
    Range, VRange, MCRange, MCVRange
```

"Functions"

- LoKi offers about >100 "Functions":
- "Particle Functions", e.g.

C++ type

alias

```
LoKi::Particles::Momentum             P
LoKi::Particles::Identifier           ID
LoKi::Vertices::ImpactParameter       IP
```

- "Vertex Functions"

```
LoKi::Vertices::VertexChi2            VCHI2
```

- "MCParticle Functions"

```
LoKi::MCParticles::ProperLifeTime   MCTIME
```

- "MCVertex Functions"

```
LoKi::McVertices::MCVertexDistance MCVDIST
```

Metafunctions (~20)

- Transverse momentum of the first daughter

```
CHILD( PT , 1 )
```

- $\Delta_{LL}(K-\pi)$ for the first daughter of the first doughter

```
CHILD( CHILD( PIDK , 1 )  , 1 )
```

- Minimal $\Delta_{LL}(K-\pi)$ for all daughter kaons in the decay tree:

```
MINTREE( PIDK , "K-" == ABSID )
```

- And a lot of "adapters":

```
VXFUN, MCMOTH, FILTER, …
```

Functions & Cuts

- Operations with functions:

```
Fun fun = P + PT / GeV * sin( 1/ M ) ;
Fun fun = pow(P,Q) + atan2(PX,PY);
```

- Comparisons:

```
Cut cut =  PT > 1.5 * GeV ;
```

- Boolean operations

```
Cut cut = ( PT > 1.5 * GeV )&&( Q < 0 ) ;
```

- Special cases (ID, ABSID, MCID, MCABSID):

```
Cut cut = “pi+” ==    ID ;
Cut cut = “mu-” == ABSID ;
```

Every day idioms: simple selections

```
#include “LoKi/LoKi.h”
LOKI_ALGORITHM( MyAlg)
{
  using namespace LoKi       ;
  using namespace LoKi::Cuts ;
  Range pions = select( “pi” ,
       “pi+” == ABSID && PT > 0.5 * GeV ) ;
  info() << “ found pions:” << pions.size()
         << endreq ;
  return StatusCode::SUCCESS ;
};
```

Select from all loaded/created particles

TAG

Cuts: π^+ and π^- with p_T>500 MeV/c

Simple selections (II)

- Select from other selected range :

```
Range pions = select( “pi” , “pi-” == ABSID ) ;
Range pos   = select( “pi+” , pions , Q > 0 ) ;
```

- Select from `KeyedContainer`:

```
const Particles* p =
    get<Particles>(“Phys/MyChannel/Particles”);
Range bs = select( “myBs0” , p ,
                   “B_s0” == ID );
```

- Select from arbitrary sequence `seq` :

```
Range k0s = select( “myK0S” ,
      seq.begin() , seq.end() , “KS0” == ID );
```

Trivial 1-particle loops

- Nothing special: **Range** behaves like **STL**-container

```
Range pions = select( … )   ;
for( Range::iterator ipi = pions.begin() ;
                pions.end() != ipi ; ++ipi )
{
 const Particle* p = *ipi ;
  info() << “ pion momentum:”
         << P( p ) / GeV << endreq
};
```

Multiparticle loops:

- Loop over selected particle tags:

```
Range mypi = select( “myPi+” , … );
Range myK  = select( “myK-” , … );
for ( Loop D0 = loop( “myK- myPi+” , “D0” ) ;
                        D0 ; ++D0 )
{
  plot( M( D0 )/GeV,“K pi m”,1.5,2.0 );
  if  ( VCHI2( D0 ) > 100 ) { continue ; }
  plot( M( D0 )/GeV,“K pi m,chi2”,1.5,2.0);
}
```

Loop objects behaves as **Particle**

Loop objects behaves as **Vertex**

Fits

- Different fitting strategies:
- In the loop declaration:

```
for( Loop D0 = loop( “myK- myPi+” , “D0” , FIT )
```

- here **FIT** =

```
FitVertex          (Default)
FitMassVertex
```

- In the loop body:

```
for ( Loop D0 = … ; D0 ; ++D0 )
{
 StatusCode sc = D0->fit( FIT ) ;
}
```

```
FitVertex
FitMassVertex
FitDirection
FitLifeTime
```

```
Fit1 && Fit2 && Fit3
```

Save something interesting

```
Cut cut = … ;
for ( Loop D0 = … ; D0 ; ++D0 )
{
 if ( !cut( D0 ) ) { continue ;}
 D0->save( “myD0” ) ;
}
```

TAG

- Extract saved particles:

```
Range d0 = selected( “myD0” )
info() << “ D0 saved: ”
       <<   d0.size() << endreq;
```

Get something “working” (I)

```
Range mu = select(“mu” , “mu+” == ABSID &&
  PIDmu > -2 && PT > 500 * MeV ) ;


Cut dm = ADMASS(“J/psi(1S)”) < 100 * MeV ;
for( Loop Jpsi = loop( “mu mu”,“J/psi(1S)” );
                         Jpsi ; ++Jpsi )
{
  if ( 0 != SUMQ(Jpsi) ||
        VCHI2(Jpsi) > 100 ) { continue ; }
  if ( dm( Jpsi) ) { Jpsi->save(“psi”) ; }
};
```

$\Sigma q = 0$ and $\chi^2 < 100$

Get something “working” (II)

```
Range K = select(“K” ,
     “K+” == ABSID && PIDK > 0 ) ;

Cut dm = ADMASS(“phi(1020)”) < 12 * MeV ;
for( Loop phi = loop( “K K”,“J/psi(1S)” );
                        phi ; ++phi )
{
  if ( 0 != SUMQ(phi) ||
        VCHI2(phi) > 100 ) { continue ; }
  if ( dm( phi) ) { phi->save(“phi”) ; }
};
```

$\Sigma q = 0$ and χ^2<100

Get something “working” (III)

```
Cut dm = ADMASS(“B_s0”) < 500 * MeV ;
for( Loop Bs = loop( “psi phi”,“B_s0” );
                              Bs ; ++Bs )
{
  if ( VCHI2(Bs) > 100 ) { continue ; }
  if ( dm( phi) ) { Bs->save(“Bs”) ; }
};
Range Bs = selected(“Bs”);
if( !Bs.empty() ){ setFilterPassed( true ) ;}
```

$\Sigma q = 0$ and $\chi^2 < 100$

Or everything together:

```
Range mu = select(“mu” , “mu+” == ABSID && PIDmu > -2 &&
  PT > 500 * MeV ) ;
Range K = select(“K” , “K+” == ABSID && PIDK > 0 ) ;
Cut  dmPsi = ADMASS(“J/psi(1S)”) < 100 * MeV ;
Cut  dmPhi = ADMASS(“phi(1020)”) <  12 * MeV ;
Cut  dmBs  = ADMASS(“B_s0”)      < 500 * MeV ;
Cut  q     = 0 == SUMQ   ;
VCut chi2 = VCHI2 < 100 ;
pattern(“psi”, “mu mu”,“J/psi(1S)”, dmPsi && q , chi2 );
pattern(“phi”, “K K”  ,“phi(1020” , dmPhi && q , chi2 );
pattern(“Bs” , “psi phi” , “B_s0” , dmBs       , chi2 );
Range Bs = selected(“Bs”) ;
if( !Bs.empty() ) { setFilterPassed(true);}
```

1 page !!!

MC match

- `LoKi` uses own concept of MC-truth matching, described in details in LUG
  - "Loose" matching: none relations can be lost ☺
  - Some "extra" relations could be a bit confusing ☹
  - Technically based on `Relation Tables` from `Kernel/Relations` package
    - Requires:

```
IRelation<ProtoParticle,MCParticle,double>
IRelation<Particle,MCParticle>
IRelation<Particle,MCParticle,double>
```

No way for more or less smooth transition to `Linkers`

- Natural coupling with MCDecayFinder tool and `MCParticle` selections
- Few helper adapter functions

MCMatch

```
MCMatch mc = mctruth() ;
MCRange mcPsi = mc-> findDecay(
   “B_s0 -> ^J/psi(1S) phi(1020) ”);

Cut truePsi = MCTRUTH( mc , mcPsi ) ;
For ( Loop Jpsi = loop(“mu mu”, … ) ;
  Jpsi ; ++Jpsi)
{
    if( !truePsi( Jpsi) ) { continue ; }
}
```

Evaluates to `true`, if both muons come from true MC J/psi from this decay chain

MC truth Match

```
Cut truePsi = MCTRUTH( mc , mcPsi ) ;
Cut truePhi = MCTRUTH( mc , mcPhi ) ;
Cut trueBs  = MCTRUTH( mc , mcBs ) ;
Cut trueMu  = MCTRUTH( mc , mcMu ) ;
Cut trueK   = MCTRUTH( mc ,  mcK  ) ;
For( Loop Bs = loop(“psi phi”, … );Bs;++Bs)
{
tuple -> column(“mcbs” ,trueBs  (Bs   ) );
tuple -> column(“mcpsi”,truePsi (Bs(1)) );
tuple -> column(“mcphi”,truePhi (Bs(2)) );
tuple -> …
}
```

Select tracks with $\min(\chi^2)_{IP} > 25$

- Very efficient operation if done **BEFORE** looping, the combinatorics is reduced significantly (and **huge** gain in CPU!)

Vertices are selected in a similar way

```
VRange pvs = vselect( “PVs” ,
               Vertex::Primary == VTYPE ) ;
```

The function objects itself

```
Cut mips = MIPCHI2( geo() , pvs ) > 25 ;
Range pions = select( “pi” ,
               “pi+” = ABSID && mips ) ;
```

Select pions not from primary verstices

Select Primary vertes

- Select primary vertex according to some criteria, e.g. the vertex with minimal χ^2_{IP}:

```
VRange pvs = vselected(“PVs” ,
                Vertex::Primary == VTYPE ) ;
For ( Loop Bs = loop(“psi phi”,…);Bs;++Bs)
{
  const Vertex* pv =
    SelectPrimaryVertexMin(
    pvs.begin() , pvs.end() ,
    VIPCHI2( Bs , geo() ) ,
    VPSD( Bs, geo() ) > -200 * micrometer ) ;
}
```

Sequence of vertices

Selection criterion

Cut: B_s should not be “too” upstream with respect to selected primary vertex

Other examples

- “Pedagogical”

  `Ex/LoKiExample` package

- “Realistic”

  `PhysSel/B2XGamma`

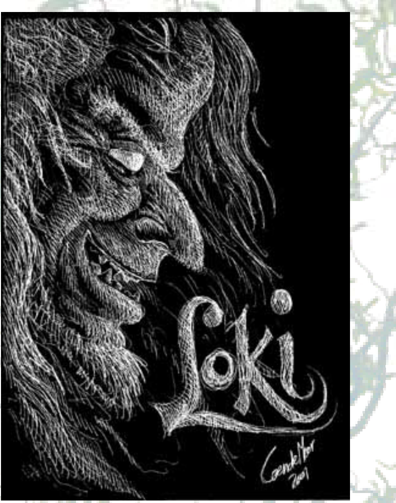
  `PhysSel/Bs2PhiPhi`

  `PhysSel/B2DstarX2D02hh`

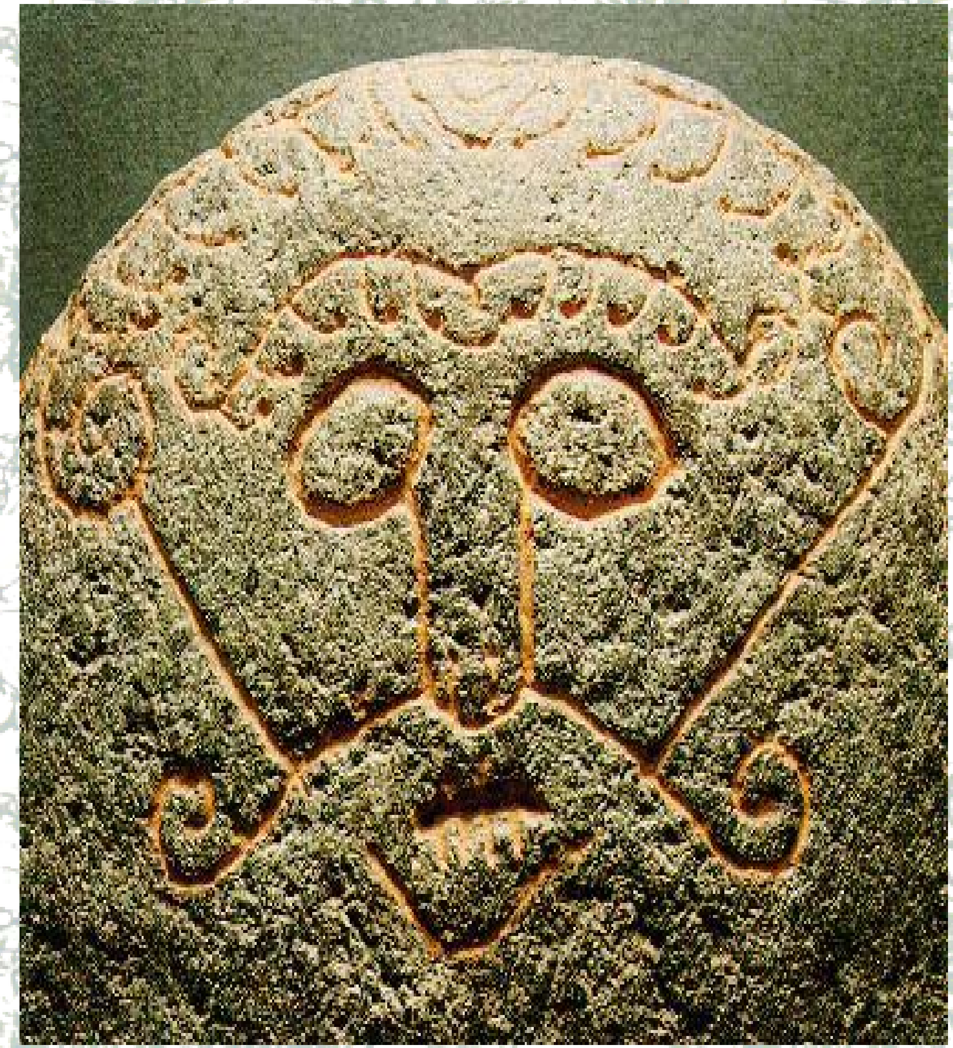
- There is a lot of code fragments in LUG
- A lot of examples can be found through the archive of lhcb-loki@cern.ch mailing list
- My office is 1-R-010

LoKi I

- Loki is a god of wit and mischief in Norse mythology
- **Lo**ops & **Ki**nematics



LoKi II



LoKi III