DTags and the $D$ Skims

Peter Onyisi

CLEO 101 Day 7
15 June 2006
A lot of CLEO-c physics involves $D$ mesons!

- We want to agree on how we reconstruct $D$ candidates: simpler for the users, less error-prone, systematics only need to be done once.
- We have a wonderful combinatorics system in DChain, as you’ve seen, to do the hard work for us.

Our tools are:

- DTags, which are software objects encapsulating a $D$ decay;
- the $D$ skims, which provide standard datasets and $D$ selection requirements.

Documentation:

- D Tagging Home Page
- D Tagging Hypernews
- Bo Xin’s $D$ Skim Status Page
- CBXes on cuts in skims: Unpublished “version 1”, CBX 06-11 (“version 2”)
DTags are DChain objects (subclassed from CDDecay)

You do not make them yourself: we have a Producer to do that for you, and tcl scripts to set up the producer properly

As usual for DChain, charge conjugates are included automatically

Each DTag has a mode number: e.g. $D^0 \rightarrow K^-\pi^+$ is mode 0, $D^+ \rightarrow K^-\pi^+\pi^+$ is mode 200

The mode numbers are given symbolic names in DTag.h

Each mode also has an official name that you use to actually extract it from the Frame; e.g. $D^0 \rightarrow K^-\pi^+$ is “D02K-Pi+”, $D^+ \rightarrow K^-\pi^+\pi^+$ is “D+2K-Pi+Pi+”

The mode names are listed in DTagDecayModes.cc

You extract the DTagList for a given mode name from the Frame, then iterate to get individual $D$ candidates
The class DTag is smarter than your average CDDecay:

- It stores the mode number and mode name (decayMode() and name())
- charm() tells you if it had a $c$ or a $\bar{c}$ (through the overall charge or the sign of the kaon)
- Kinematic information is already stored in them: you can call beamConstrainedMass() ($= \sqrt{E_{beam}^2 - |\vec{p}_{cand}|^2}$) and deltaE() ($= E_{cand} - E_{beam}$)
- Functions let you access the tracks and showers not used in making the candidate (the “other side”)
Example DTag Extraction

```c++
#include "DTag/DTagList.h"
...
// extract one mode
FAItem<DTagList> tagList;
extract(iFrame.record(Stream::kEvent), tagList, "D02K-Pi+");
DTagList::const_iterator tagListEnd = (*tagList).particle_end();
for (DTagList::const_iterator tagListItr = (*tagList).particle_begin();
    tagListItr != tagListEnd;
    ++tagListItr) {
    const DTag& tag = (*tagListItr).particle();
    // do with it what you will
    report(DEBUG, kFacilityString)
       << "Mode name, charm: " << tag.name() << " "
       << tag.charm() << endl;
}
```

(mkproc -dtag will generate a skeleton for you.)

Now: how do you get DTags into the Frame in the first place?
We have already made the DTags for you! (At least, for data and generic Monte Carlo.)

- These tags are put in eventstore:

```plaintext
module sel EventStoreModule
eventstore in 20060301 dtag all dataset data31
...
```

- Only events with one or more DTags found are included in the skim! In particular, to get efficiency denominators from Monte Carlo you have to run on the full sample! (That’s why it’s called a skim, after all.)

- You have access to all the reconstruction information in the physics grade as well.

- The details of what releases were used are here.
You may well generate your own Monte Carlo — you then need an extra step to get DTag objects.

- This is an unfortunately complicated topic.
- Datasets data31–37 \((E_{cm} = 3.77 \text{ GeV})\) and data38–40 (higher energy) use different methods ("version 1" and "version 2")
- At the current time:
  - data31-37 (version 1) MC should be skimmed in release 20050316 FULL 1 using the instructions here;
  - data38-40 (version 2) MC should be skimmed in release 20060224 FULL 1 using the instructions here.
- The version 1 instructions create a new pds file which you can then run your processor on. I recommend using a more recent release when running your processor.
- The version 2 instructions run your processor while doing the skim. You are tied to the 20060224 FULL 1 release.
DDoubleTag Objects

You may wish to find double tags — where you reconstruct both the \( D \) and the \( \bar{D} \). The recommended method of doing this is in a recent release (20060224_FULL_2 or later):

```
#include "DDoubleTag/DDoubleTagList.h"
...
// extract() tagSideList with the mode you want for clean tag
FAItem< DTagList > tagSideList;
extract( iframe.record(Stream::kEvent), tagSideList, "D02K-Pi+" );

// extract() signalSideList with the mode you want for signal side
FAItem< DTagList > signalSideList;
extract( iframe.record(Stream::kEvent), signalSideList, "D02KsPi0Pi0" );

// Form combination with the magic of DChain
DDoubleTagList doubleTags;
doubleTags = (*signalSideList) * (tagSideList->bar());
```

(again, mkproc -ddoubletag will generate a skeleton for you.)

Use taggedD() and taggedDbar() to get the DTags in the DDoubleTag — **but beware**, they’re really the first and second \( D \) in the combination.
You can create a DTagList combining several modes together:

```cpp
STL_VECTOR(DTag::DecayMode) modes;
modes.push_back(DTag::kD0toKPi);
modes.push_back(DTag::kD0toKPiPi0);
modes.push_back(DTag::kD0toKPiPiPi);

DTagList allTagList;
for (STL_VECTOR(DTag::DecayMode)::iterator mode = modes.begin();
    mode != modes.end();
    ++mode) {
    extract(iFrame.record(Stream::kEvent), tagList,
            DTag::modes().name(*mode));
    allTagList += (*tagList);
}
// iterate over allTagList
```

You can use allTagList like the lists returned from the extract() call (except you have to use DTagList::iterator instead of DTagList::const_iterator)
Instead of using your own vector, you can iterate over, e.g., all $D^+$ modes by using the iterators from `DTag::modes()`:

```cpp
DTagList allTagList;
for (DTagDecayModes::const_iterator mode = DTag::modes().Dp_begin();
    mode != DTag::modes().Dp_end();
    ++mode) {
    extract(iFrame.record(Stream::kEvent), tagList,
            (*mode).second.name());
    allTagList += (*tagList);
}
// iterate over allTagList
```

(Recommended for exploration ONLY...)

Tricks of the Trade II

Peter Onyisi

DTags and the $D$ Skims
Each mode has different $\Delta E$ and $m_{BC}$ resolution — estimates of the $1\sigma$ widths can be obtained in the code:

```c++
#include "DTag/DTagModeWidth.h"
...
FAItem<DTagModeWidth> aDTagModeWidth;
extract(iFrame.record(DTagStream::kModeWidth), aDTagModeWidth);

if( d.deltaE() > (0 - 3*aDTagModeWidth->sigmaDeltaELeft( modeNum )) &&
   d.deltaE() < (0 + 3*aDTagModeWidth->sigmaDeltaERight( modeNum ))) {
   // Candidate passes 3 sigma delta E cut
   ...
}
```
Since DTagLists and DDoubleTagLists are DChain lists, you can use selection functions to apply cuts:

```cpp
...  
DChainBoolean mySelector(DTag& iDtag) {
    return (abs(iDtag.deltaE()) < 0.035);
}
...  
STL_VECTOR(DTag::DecayMode) modes;
modes.push_back(DTag::kDptoKPiPi);
modes.push_back(DTag::kDptoKPiPiPi0);

DTagList tagSideList(&locSelector);
FAItem<DTagList> tagList;
for (STL_VECTOR(DTag::DecayMode)::iterator mode = modes.begin();
    mode != modes.end();
    ++mode) {
    extract(iFrame.record(Stream::kEvent), tagList,
        DTag::modes().name(*mode));
    tagSideList += (*tagList);
}
```
Exercise

- Create a processor that extracts $D^0 \rightarrow K^-\pi^+$, $K^-\pi^+\pi^0$, and $K^-\pi^+\pi^-\pi^+$ tags and plots $m_{BC}$ and $\Delta E$ for them.
- Establish the existence of $D^+ \rightarrow \pi^+\eta$ by using double tag events with the $D^-$ going into tag modes of your choice (hint: your choice should be mode numbers 200–205). Include charge conjugate events, of course.
- Skim the signal Monte Carlo you’ve generated.
Advanced Topics
(for your future edification)
There’s nothing magical about the cuts we use for the $D$ candidates — we could make a different choice.

To study the effects of the cuts, you can use alternative skims (“NoPID”, “LOOSE”)

You get these by using “production tags” in the `extract()` call

The default skim (production tag “”) is always available

<table>
<thead>
<tr>
<th></th>
<th>NoPID</th>
<th>LOOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1 data</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Version 1 generic MC</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Version 2 data</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Version 2 generic MC</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

Where NoPID not available, `setup_dtag_selection` can make it from LOOSE.

Where LOOSE not available, you need to run the skim yourself.
Tightening Cuts

- We have a tool that starts with a loose skim and tightens it in ways you specify
  - For example, it can tighten the LOOSE skim down to the NoPID skim
- Only available in releases with the version 2 code
- It is accessed with the tcl command `setup_dtag_selection`
- Further details in [CBX 06-11](#)