

# TMBAnalyze HOWTO

Reiner Hauser, [rhauser@fnal.gov](mailto:rhauser@fnal.gov) <<mailto:rhauser@fnal.gov>>

v1.2, 16 June 2005

This document describes how to use `tmb_analyze` to create TMBTrees. It is meant for releases p16.05.01 or newer. Please report any mistakes, typos etc. back to *me* <<mailto:rhauser@fnal.gov>> .

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Getting Started</b>	<b>2</b>
2.1	Which Release to use . . . . .	2
2.2	Setting up the Release . . . . .	2
<b>3</b>	<b>Running TMBAnalyze_x</b>	<b>2</b>
3.1	Using <code>d0tools</code> . . . . .	2
3.2	Running it on local files . . . . .	3
3.3	Using SAM on ClueD0 . . . . .	3
3.4	Using SAM on CAB . . . . .	3
3.4.1	Output Directories . . . . .	4
3.4.2	Multiple Jobs . . . . .	4
3.4.3	Number of Input Files . . . . .	4
<b>4</b>	<b>Production Mode Tree Generation</b>	<b>4</b>
4.1	Datasets . . . . .	4
4.2	Recursive Datasets . . . . .	5
4.2.1	<code>run-skim.sh</code> . . . . .	7
<b>5</b>	<b>Storing Trees into SAM</b>	<b>9</b>
<b>6</b>	<b>Customizing TMBAnalyze</b>	<b>10</b>
<b>7</b>	<b>Combining TMBAnalyze with Skimming</b>	<b>10</b>
<b>8</b>	<b>Merging Output Files</b>	<b>11</b>
8.1	Merging Root Files . . . . .	12
8.2	Merging Metadata Files . . . . .	13

8.3 Putting it Together . . . . .	15
<b>9 Running Over Monte Carlo Files</b>	<b>16</b>
9.1 Running over pure Monte Carlo Files . . . . .	17
<b>10 Extending TMBAnalyze</b>	<b>17</b>
10.1 Storing TClonesArrays . . . . .	18
10.2 Storing Single Objects . . . . .	19
10.3 Storing other Persistent Information . . . . .	19

## 1 Introduction

This document describes how to run `TMBAnalyze` with `d0correct` in p16.05.01 and newer releases. This includes generating the correct metadata for SAM, storing trees back into SAM and using `TMBAnalyze` with additional filters for skimming. It also discusses how to extend `TMBAnalyze` with your own classes.

## 2 Getting Started

### 2.1 Which Release to use

For pass 1 data you can use/should release p16.04.00. For pass 2 data you *must* use p16.05.01 or newer. `d0correct` was not certified in any older release for pass 2 data.

### 2.2 Setting up the Release

```
% setup D0RunII p16.05.01
% newrel -t p16.05.01 work
% cd work
% d0setwa
% setup d0tools
```

## 3 Running TMBAnalyze\_x

### 3.1 Using d0tools

The preferred way to run `TMBAnalyze` is with `d0tools`. Since the the default setup now includes `d0correct` by default you need additional files to be available at run-time. `d0tools` takes care of all of this.

To run it in stand-alone mode, check the `$DOTOOLS_BIN/runTMBAnalyze` script to see what additional parameters it passes to `rund0exe`.

`runTMBAnalyze` is a wrapper script around the generic `rund0exe` script. You can just call it after you have setup `d0tools`.

Run `runTMBAnalyze -h` to get a help message.

### 3.2 Running it on local files

For local files you should provide a list of input files and pass it via the `filelist=` argument to `runTMBAnalyze`

```
runTMBAnalyze -filelist=myfiles
```

where `myfiles` should contain the absolute pathnames to your input files.

### 3.3 Using SAM on ClueD0

You can use `TMBAnalyze` with SAM on `clued0`, but *only* in batch mode. The way to submit a batch job is to simply add the `-batch` option and additional parameters for cpu time and memory consumption. In addition you have to specify the SAM definition name.

```
runTMBAnalyze -defname=MySAMDefinition -batch -cput=8:00:00 -mem=600mb
```

You should use this only for testing, not for large production jobs.

### 3.4 Using SAM on CAB

This is the preferred way to run production jobs. You can use all the usual options for D0 executables. We describe the most common here and recommend some good practices.

You need a *scratch* directory into which `d0tools` will store a tar file containing your current working directory (minus any stuff which is not needed for running). It is therefore preferable if you run an executable directly out of a release instead of compiling it yourself. Even if you need to change RCP parameters, there is no need to rebuild the executable.

The scratch directory can *not* be a subdirectory of your working area. `d0tools` will refuse to use it if you try it. Instead it should point somewhere to a `/work` or `/rooms` area where you have disk space available.

The two CAB systems are called `cabsrv1` and `cabsrv2`.

```
runTMBAnalyze -defname=MySAMDefinition -cabsrv1 \  
-scratch=/work/mymachine-clued0/myname/scratch [ ..other options.. ]
```

Once you have let `d0tools` create such a tar file, you can just re-use it. However, whenever you change an RCP parameter, you *must* delete the old one or re-create a new one with a different name.

```
runTMBAnalyze -defname=MySAMDefinition -cabsrv1 \  
-scratch=/work/mymachine-clued0/myname/scratch \  
-cabtar=mycabjob.tar [ ...other options... ]
```

### 3.4.1 Output Directories

By default the results of your job will be returned in the scratch directory. Instead of copying them afterwards by hand to their final destination, you can instruct `d0tools` to directly store them there. You do this via two options:

```
runTMBAnalyze -defname=MySAMDefinition -cabsrv1 \  
  -scratch=/work/mymachine-clued0/myname/scratch \  
  -cabtar=mycabjob.tar [ ...other options... ] \  
  -cabouthost=servermachine-clued0 -caboutpath=/rooms/project/dir
```

You should always specify the correct output host name for performance reason. Otherwise there is a lot of unnecessary copying across the network. This is especially true if your output directory is on one of the `/prj_root/...` disks. These are located at the Feynman Computing Center, so specifying the output path but not the host would lead to a copy from FCC to D0 and then back again.

You can always find the host name by typing `df /my/output/path`. For the project disks it should be typically something called `d0srvNUMBER`.

### 3.4.2 Multiple Jobs

If you run over a large data set, you can use multiple parallel jobs. Just specify the `-jobs=` option. The maximum number of parallel jobs is 20. This only works for CAB.

```
runTMBAnalyze -defname=MySAMDefinition -cabsrv1 \  
  -scratch=/work/mymachine-clued0/myname/scratch \  
  -jobs=20 [...other options.. ]
```

### 3.4.3 Number of Input Files

The default setup will create one output root file for one input thumbnail file. If you use SAM and want to store the results back into SAM you should not try to create one big output file. ROOT will open by itself new output files if the first one grows to big, but since we have no control over that, all the SAM parentage information will be confused.

If you run over a smaller sample and don't care about this, you can change in `tmb_tree_maker/rcp/TMBTree.rcp` the `InputFilesPerFile` variable and set it either to 0 (all input goes into one output file) or a number higher than 1. For instance, if you do some additional filtering, you may know that it is safe to have about 4 input files per output file due to the reduced event count.

## 4 Production Mode Tree Generation

### 4.1 Datasets

You can find the Pass 2 dataset definitions on the Common Samples Group page. They all follow this pattern:

```
CSskim- $\{\text{SKIM}\}$ -PASS2- $\{\text{RECO}\}$ 
```

where  $\{\text{SKIM}\}$  is the name of the skim and  $\{\text{RECO}\}$  is the version which was used to reconstruct the events. If you don't care about the reco version, you can also use a definition that encompasses all events for a skim: `CSskim- $\{\text{SKIM}\}$ -PASS2`.

## 4.2 Recursive Datasets

The datasets can be quite large (hundreds of files), so you have to process them with many jobs. Instead of splitting the dataset by hand, you should use SAM to do the book-keeping for you.

The basic idea is to define a new dataset definition based on the original one, minus all the files that you already processed. Remember that a definition is evaluated every time you make a new snapshot (i.e. start a new project on it).

To make this more concrete, here is an example (myname is supposed to be your login name):

```
sam create definition \
  --group=dzero \
  --defname=myname_cskim_tree_v1_2MU_p14.06.00 \
  --dim="__set__ CSskim-2MU-PASS2-p14.06.00 minus \
    ((project_name myname_cskim_tree_v1_2MU_p14.06.00-% \
      and consumed_status consumed) and consumer myname)"
```

The idea is the following:

- We define a dataset based on a common skim definition.
- We force all our projects that we run on this new definition to have a certain name.
- The definition is the original skim minus all files that have been processed by any project with a given name pattern (note the wildcard character).

Normally d0tools will choose a project name for you (because you don't care about the details), but if the  `$\{\text{SAM\_PROJECT}\}$`  environment variable is defined, it will use that instead. We have to make sure that it is unique, though.

The way to do a production skim is now:

```
setenv SAM_PROJECT myname_cskim_tree_v1_2MU_p14.06.00-'date +%Y%m%d-%H%M%S'
runTMBAnalyze -defname=myname_cskim_tree_v1_2MU_p14.06.00 -jobs=20 [ ..other options...]
```

You have to wait until these twenty jobs are finished before you submit the next one ! SAM will coordinate file transfer to the parallel jobs and make sure there are no duplicates, but it won't do this between different projects.

While the jobs are running you can occasionally check the definition in the web browser or on the command line. You should see that it will have less and less files over time. Once there are no files left, you are done.

```
sam translate constraints --dim="__set__ myname_cskim_tree_v1_2MU_p14.06.00"
```

The restriction to have only 20 jobs running is not a big one: usually you have one definition per reco release and they are independent, so you can submit 20 jobs for each.

At the end you should check that you have processed all files. Sometimes things go wrong, be it with SAM, CAB, the network or the output servers or disks. So you should compare the file list of the original definition with all the files you processed. You can find this information in the `events.read` file in each output directory.

```
grep Input /path/to/output/dirs/*/events.read
```

Here is a little script to do that for both files and events:

```
#!/bin/bash
#
# usage:
#
#   count.sh <dirlist>...
#
# Count number of input files and events processed.
# <dirlist> should be the list of batch output directories
#

# $* = list of files
count_events()
{
    grep 'Total events:' $* | (
        x=0
        while read a b count
        do
            x='expr ${x} + ${count}'
        done
        echo "Event count = $x"
    )
}

# $* = list of files
count_files()
{
    grep 'Input file:' $* | grep -v 'END OF STREAM' | (
        c=0
        while read a b filename
        do
            c='expr ${c} + 1'
        done
        echo "File count = $c"
    )
}
```

```

check_dir()
{
    for d in $*
    do
        if [ -r ${d}/events.read ]; then
            files="${files} ${d}/events.read"
        else
            echo "No events.read file in ${d}"
        fi
    done
}

files=""
check_dir $*
count_events $files
count_files $files

```

If the number of files and events don't correspond to what SAM tells you, you missed some... However, the latest version of `d0correct` removes duplicate events by default, so the number of events might not be completely identical. Check the log files of your jobs (`TMBAnalyze.out`) for a line like this:

```

duplicated events rejected          181

```

The sum of the rejected events plus what you have in your output files should match the number of events in the input dataset.

In many cases it's just a few files, you should just make a new SAM definition with exactly the missing filenames and submit a recovery job.

#### 4.2.1 run-skim.sh

Here is a complete script to start a tree generation for a given skim. Please read the comments and modify it before use.

```

#!/bin/sh
#
# Usage:
#
#   run-skim.sh <skim-name> <release> [ <version> ]
#
# This assumes that the dataset definition you want to use is
#
#   <user>-csskim-tree-<skim>-<release><version>
#
# this definition should in turn be based on the official skim definitions
# from the common sample group. Note there is no indication about which

```

```
# tmbfixer was used here.
#
# Your dataset definition should look like this:
#
# __SET__ CSskim-2MU-PASS2-p14.06.00 minus
# ((PROJECT_NAME <user>-csskim-tree-<skim>-<release><version>-%
# and CONSUMED_STATUS consumed) and CONSUMER <user>)
#
# where
# <skim> is one of 2MU, 2EM, EMMU, etc.
#
# <release> is e.g. p14.06.00
#
# Output name will be:
#
# TMBTree-<skim>-<release>-<JOBID>
#
#
VERSION=""

# CAB system to use, set to 'srv1' or 'srv2'
CAB_SERVER="srv2"

# number of maximum parallel jobs
NUM_JOBS=20

# final output host and directory
OUT_HOST=dummy-clued0
OUT_DIR=/rooms/dummy/output

CAB_SCRATCH=/rooms/dummy/myname/scratch

# Make this non-empty for any tests you do, append ${TEST} to your dataset definition
TEST=""

usage()
{
    echo "run-skim.sh <skim-name> <release> [ version ]"
}

SKIM=""
RELEASE=""

if [ $# -lt 2 ]; then
    usage
    exit 1

```

```

fi

SKIM=$1
RELEASE=$2

if [ $# -gt 2 ]; then
    VERSION=$3
fi

BATCH_NAME=TMBTree-${SKIM}-${RELEASE}

DEFNAME=${USER}-csskim-tree-${SKIM}-${RELEASE}-${VERSION}-${TEST}
export SAM_PROJECT=${DEFNAME}-`date +%Y%m%d-%H%M%S`

if [ -f ${CAB_SCRATCH}/cabfile.tar ]; then
    TAROPTION=" -cabtar=cabfile.tar"
else
    TAROPTION=""
fi

runTMBAnalyze -name=${BATCH_NAME} -nofpe -maxopt \
    -batch -cab${CAB_SERVER} \
    -scratch=${CAB_SCRATCH} -jobs=${NUM_JOBS} \
    -defname=${DEFNAME} -cabouthost=${OUT_HOST} \
    -caboutpath=${OUT_DIR} \
    ${TAROPTION} \
    -jobname=${SKIM} \
    -fwkparams -num_files 2

```

## 5 Storing Trees into SAM

Often tmb trees are useful for other people beyond yourself or even your physics group. By storing them back into SAM, everybody with a SAM station can easily pull them out or run directly on them via SAM.

By default, TMBAnalyze will create valid metadata for your job. So after you have an output file, you can just go to the corresponding directory and do (in bash):

```

setup sam
for file in *.metadata.py
do
    sam store --descrip=${file} --source=. --dest=[...]
done

```

The destination will depend on the physics group you are in. Ask your conveners, they should be able to tell you which destination to use for root files.

If the files are on clued0, you have to do this on flotsam-clued0 or sambar-clued0. It will not work on any other machine.

If the files are on a FCC file server, you have to do it from [?don't know yet?]. Until d0mino is down, these disks are also visible from there, so it is easiest to log into d0mino and do it there.

## 6 Customizing TMBAnalyze

If you want to make changes to TMBAnalyze, there are three places to customize it. `tmb_analyze` contains the main framework RCPs. `tmb_tree_maker` contains RCPs for both controlling which branches are created and for output options. `tmb_tree_trigger_make` does the same for trigger branches.

```
addpkg tmb_analyze
addpkg tmb_tree_maker
```

If you make changes to the RCPs, remember to create a new tar file for CAB and specify the `-localrcp` and/or `-localfwkrpc` option to `runTMBAnalyze`.

In `tmb_analyze/rcp/runTMBTreeMaker.rcp` (there are variants for SAM and Monte Carlo) you can mainly add/remove packages. The default is to run with `d0correct`, including duplicate event removal.

In `tmb_tree_maker/rcp/TMBCorePkg.rcp` you can enable or disable certain branches in the root file. There is one boolean flag per branch.

In `tmb_tree_maker/rcp/TMBTreePkg.rcp` you can control the name of the output files and some other output related parameters. The filename is by default the name of the first input file plus `.root`. It allows all the options that the normal `WriteEvent` package provides.

The `Tag` option is only useful if you do additional skimming while producing the trees. See the next section.

For trigger information, you should customize `tmb_tree_trigger_maker/rcp/TMBTriggerPkg.rcp`.

## 7 Combining TMBAnalyze with Skimming

The default version of TMBAnalyze in p16.05.01 does not link against the `np_tmb_stream` package. To do custom skimming you have to relink the executable. Of course, you can provide your own filter/tagging package and include that as well.

To use the standard skimming package `np_tmb_stream`, checkout the `tmb_analyze` package and add the following lines to `bin/OBJECTS`, then `make all`.

```
RegObjectFilter
RegObjectTag
RegAndTag
```

Then add as many RCP files as you like, instantiating the `ObjectTag` package. Here is an example based on `1MU2JET_stream.rcp` from the release:

```
// MySkim.rcp

string PackageName = "ObjectTag"

string Tag = "MY_1MU2JET"

string Trigger = ( )

string Cuts = ( "Cut1" "Cut2" "Cut3" )

string Cut1 = ( "MU" "Loose==1 && PtCentral>10.0")

// tighter cut
string Cut2 = ( "JET" "Pt > 12.0 " )

string Cut3 = ( "JET" "Pt > 8.0 " )

string JetName = "JCCB"

RCP   EMid_Algo = < emreco EMReco-scone-id >
string EMid_SearchRCPs = ("clusterer","HMReco",)
RCP Muonid_Algo = <muonid MuoCandidateReco>
```

In the framework RCP add a new entry to the Packages string:

```
string Packages = "... unptmb d0corr links mytag tmb_core ..."
[...]
RCP mytag = <tmb_analyze MySkim>
```

This assumes you put `MySkim.rcp` into the `tmb_analyze` package `rcp` directory.

Now you can modify `tmb_tree_maker/rcp/TMBTreePkg.rcp` and specify your tag (or a list of tags):

```
string Tags = ( "MY_1MU2JET" )
string OutputFile = "MY_1MU2JET-%n.root"
```

Only events that have this tag will be written out.

The obvious extension is that you can create multiple tags and have multiple copies of the `TMBTreePkg.rcp` file (choosing a different output file name for each...).

Remember, you now have to run `TMBAnalyze` with the `-localbuild` option !

## 8 Merging Output Files

If you have an additional selection in your program, your output files may be very small if you left all the default parameters of `tmb_tree_maker` as they are. Typically you will increase the `InputFilesPerFile` parameter to some larger value.

As a rule of thumb, you should probably aim for output files of around 1 GB. It is not a desaster if they are only half a gigabyte or 1.5 GB, however.

Very small files are very inefficient to store on tape. Files larger than 2 GB might lead to problems with the filesystem and/or programs on Linux, depending on how you access them. A 1 GB file can still be processed by an interactive root session in a reasonable time.

If you have many small files, you should merge them into larger ones. There are two steps involved: merging the actual root file and merging the metadata.

## 8.1 Merging Root Files

Root itself provides various methods to merge two root trees. The following is a script that reads from standard input the name of the output file, followed by a list of input files. It merges all the input file into the single output file.

Typically you would use it like this:

```
echo output.root input1.root input2.root ... | root -b -l merge_root.C+
```

Here is the actual file:

```
/*
 * Merge a number of .root files into one output file.
 *
 * Execute the function like this from the command line:
 *
 * echo outputfile.root input1.root input2.root... | root -b merge_root.C+
 *
 * This macro assumes that the TTree is named 'TMBTree'. Change
 * the corresponding line below if this is different for your case.
 */

#include <string>
#include <iostream>

#include "TList.h"
#include "TFile.h"
#include "TTree.h"

// Change this line if your TTree has a different name
const char *TreeName = "TMBTree";

void merge_root()
{
    using namespace std;

    string outfile;
```

```

cin >> outfile;

TList tree_list;
std::string filename;

Int_t total_events = 0;

while(cin >> filename) {
    TFile *f = new TFile(filename.c_str());

    if(TTree *tree = (TTree *)f->Get(TreeName)) {

        cout << "Adding file: " << filename << endl;
        tree_list.Add(tree);

        total_events += (Int_t )tree->GetEntries();

    } else {
        cout << "File has no TTree named TMBTree" << endl;
    }
}

cout << "Opening output file: " << outfile << endl;
TFile output(outfile.c_str(), "RECREATE");

cout << "Merging trees...patience..." << endl;
TTree::MergeTrees(&tree_list);
output.Write();
output.Close();

cout << "Total Events: " << total_events << endl;
}

```

## 8.2 Merging Metadata Files

The following script takes the corresponding metadata files and creates a single output metadata file. You would call it like this:

```
python merge_metadata.py output.root input1.root input2.root...
```

It will write the metadata file in the same directory where the `output.root` file lies. It expects the metadata for the input files in the same directory as the `.root` files.

```
#!/usr/bin/python
#
# Merge metadata information for multiple root files.
```

```
#
# usage:
#
#   merge_metadata outputfile.root file1.root file2.root...
#
# It will write the outputfile.root.metadata.py into the same
# directory where outputfile.root is located.
#
# It expects to find the metadata.py files for the input files
# at the same location as the input *.root files.
#

import sys
import os.path

# SAM
from import_classes import *

# Python import interface
import imp

# At least the output name is required
if len(sys.argv) < 2:
    print "Usage: merge_metadata.py outputfile root.metadata.py input1.root input2.root..."
    sys.exit(1)

output_name = sys.argv[1]

#
# Initial values, we take some of the from the first file
# we encounter, like 'tier', 'start_time', 'end_time', 'pid'
#
sizeK = 0
first = 0
last = 0
events = 0
start_time = ''
end_time = ''
pid = 0
parents = []
tier = 'unknown'

for f in sys.argv[2:]:
    m = imp.load_source("meta", f + ".metadata.py", open(f + ".metadata.py"))
    if sizeK == 0:
        first = m.TheFile.events.begin
        last = m.TheFile.events.end
```

```

pid    = m.TheFile.pid

start_time = m.TheFile.start_time
end_time   = m.TheFile.end_time
tier       = m.TheFile.tier

else:
    if tier != m.TheFile.tier:
        print "Different data tiers: expected %s, got %s" % ( tier, m.TheFile.tier )

sizeK += m.TheFile.sizeK
events += m.TheFile.events.num

if m.TheFile.events.begin < first:
    first = m.TheFile.events.begin

if m.TheFile.events.end > last:
    last = m.TheFile.events.end

parents.extend(m.TheFile.parents)

#
# Determine real file size, we don't really trust 'sizeK', since there
# will be a different number of ROOT headers etc. per file
#
root_outfile = open(output_name)
root_outfile.seek(0,2)
real_sizeK = root_outfile.tell()/1024 + 1

outfile = open(output_name + ".metadata.py", "w")

print >>outfile,"from import_classes import *"
print >>outfile,"TheFile = ProcessedFile("
print >>outfile,"    name = '%s'," % output_name
print >>outfile,"    sizeK = %d," % real_sizeK
print >>outfile,"    events = Events(%d, %d, %d)," % ( first, last, events)
print >>outfile,"    stream = '',"
print >>outfile,"    tier = '%s'," % tier
print >>outfile,"    start_time = '%s'," % start_time
print >>outfile,"    end_time = '%s'," % end_time
print >>outfile,"    pid = %d," % pid
print >>outfile,"    parents = ", parents, ")"

```

### 8.3 Putting it Together

Finally, the following script `merge_trees.sh` will call the two scripts described above together:

```

#!/bin/bash
#
# usage:
#   merge_trees.sh output.root [ file.root ]+
#
# Merges the ROOT trees in the given files into one output file.
#
# It expects to find a 'file.root.metadata.py' file in the same
# directory as the 'file.root' file and will create a new
# metadata file for the merged trees, 'output.root.metadata.py'
#

usage()
{
    echo "usage: $0 output.root [ input.root ]+"
}

if [ $# -lt 2 ]; then
    usage
    exit 1
fi

outfile=$1
shift

for file in $*
do
    if [ ! -f ${file} ]; then
        echo "Cannot find file: ${file}"
        exit 1
    fi
    if [ ! -f ${file}.metadata.py ]; then
        echo "Cannot find metadata for ${file}: ${file}.metadata.py"
        exit 1
    fi
done

echo ${outfile} $* | root -l -b merge_root.C+
python merge_metadata.py ${outfile} $*

```

## 9 Running Over Monte Carlo Files

If the Monte Carlo files have been reconstructed, you can run `TMBAnalyze_x` as usual on them, except that you should select the proper RCP file. There are variations `runTMBTreeMaker_MC.rcp` for running over local files, as well as a version for SAM, `runTMBTreeMakerSAM_MC.rcp`.

To apply smearing to the Monte Carlo, use the `runTMBTreeMaker_MCSmear.rcp` file instead.

## 9.1 Running over pure Monte Carlo Files

If you have a Monte Carlo file without `d0reco` thumbnails, these standard RCP files will not work. Instead, you should use a modified version as shown below. This will only produce the various Monte Carlo particles, vertices and event infos and nothing else in the output tree.

```
string InterfaceName = "process"
string Packages = "geo read config tmb_core tmb_mc tmb_tree dump"

RCP geo      = <geometry_management geometry_management>
RCP read     = <d0reco D0recoReadEvent>
RCP config   = <run_config_fwkc RunConfigPkg>
RCP tmb_core = <tmb_tree_maker TMBCorePkg>
RCP tmb_mc   = <mc_analyze TMBTreeMCPkg>
RCP tmb_refs = <tmb_analyze TMBRefsPkg>
RCP tmb_tree = <tmb_tree_maker TMBTreePkg>

RCP dump     = <thumbnail tmbDumpEvent>

int DumpPeriod = 1
```

## 10 Extending TMBAnalyze

You can extend `TMBAnalyze_x` without any modifications to the original packages. This is useful if you want to add new branches or your own private information while generating the `TMBTree`. This section assumes that you have basic knowledge about ROOT's persistency system.

As a first step, you should create your own CVS and framework package where you will put your code. In the following we assume that your framework class is called `TMBExtendPkg`. This class only has to inherit from `fwk::Package` base class:

```
#include "framework/Package.hpp"
#include "tmb_tree_maker/TMBMaker.hpp"

namespace MyExtension {

class TMBExtendPkg : public fwk::package {
public:
    TMBExtendPkg(fwk::Context *ctx);
    std::string packageName() const;
    static const std::string package_name();
    static const std::string version();
private:
```

```

        TMBMaker *_maker;
    };
}

```

Extending TMBAnalyze is very easy if your own data is either:

- a single object of a user defined class.
- a TClonesArray of multiple objects of the same class.

In both cases you should create a subclass of the TMBMaker class. We assume you have defined your own root class, e.g.

```

class MyClass : public TObject {
public:
    MyClass();
private:
    Int_t data;
public:
    ClassDef(MyClass, 1);
};

```

### 10.1 Storing TClonesArrays

We handle the case of a TClonesArray first.

Your maker class should look like this:

```

class MyArrayMaker : public TMBMaker {
public:
    MyArrayMaker(const char *name, const char *title);
    ~MyArrayMaker();
    Int_t Make(edm::Event& event);
};

```

In the constructor, you specify your branch name and pass an instance of a properly initialized TClonesArray to the base class:

```

MyArrayMaker::MyArrayMaker(const char *name, const char *title)
    : TMBMaker(name, title)
{
    _Fruits      = new TClonesArray("MyClass", 2, kFALSE);
    _BranchName = "MyArray";
    Save();
}

```

Then, in the Make() method, you actually fill the array with any data you like. The assumption is that you either get it from the edm::Event or generate it yourself.

```

Int_t MyArrayMaker::Make(const edm::Event& event)
{
    TClonesArray& array = *(TClonesArray*)_Fruits;

    for(int i = 0; i < numObjects; i++) {
        // create your object
        new (array[i]) MyClass();
    }
    return 0;
}

```

That's it. Now all we have to do is make the `MyArrayMaker` class known to `TMBAnalyze`. We do this by simply creating an instance of the class in our framework package:

```

TMBExtendPkg::TMBExtendPkg(fwk::Context *ctx)
    : fwk::Package(ctx),
      _maker(0)
{
    bool doMyClassArray = packageRCP().getBool("doMyClassArray");
    if(doMyClassArray) _maker = new MyArrayMaker("MyClassArray", "Maker for MyClass");
}

TMBExtendPkg::~TMBExtendPkg() { delete _maker; }

```

Now you can add your `RegTMBExtendPkg` to `tmb_analyze/bin/OBJECTS` and relink the executable. Alternatively, you can create your own executable by copying over `tmb_analyze/bin/OBJECTS` into your own package `bin` directory and just add any local modifications.

## 10.2 Storing Single Objects

The modifications for storing a single object instead of a `TClonesArray` are rather simple.

- In the constructor, create an object of the desired type and assign it to `_Fruits`:

```
_Fruits = new MyClass();
```

- Override the `Clear` method and call `_Fruits->Clear()` in there.

## 10.3 Storing other Persistent Information

The scheme above assumes that you have one `Maker` class for each object or array of objects which is stored in turn in a single branch.

You don't have to follow this scheme, but you will have to do more of the work yourself in this case. If you assign a null pointer to `_Fruits` the normal procedure will not apply. However, in this case, you have to

create the branch yourself. To do this, override the `MakeBranch` method and do anything you have to do for your custom branch(es).

You can get access to the common tree like this:

```
TTree *tree = gTMBTree->Tree();
tree->Branch("branch1", ...);
tree->Branch("branch2", ...);
...
```

For instance, you could store plain old ntuples here.