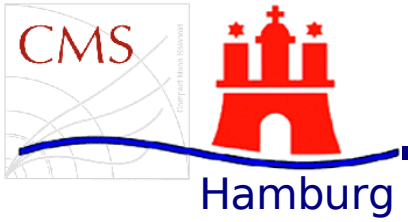

Applying Calibration Constants from the GlobalFit in CMSSW

Roger Wolf

Calibration meeting, 04.07.2008

Application of Official Calibrations

Hamburg

Get the services from:[1)]

- cvs co -r jet_corrections_CSA07_169_v3 CondFormats/JetMETObjects
- cvs co -r jet_corrections_CSA07_169_v3 JetCorrections/MCJet
- cvs co -r jet_corrections_CSA07_169_v3 JetCorrections/Modules

- uncorrected Jets
  - L2Service
  - L2Correction
- L2 corrected Jets
  - L3Service
  - L3Correction
- L3 corrected Jets
  - ...Service
  - ...Correction

```
es_source L3JetCorrectorFKt4 = L3AbsoluteCorrectionService {
string tagName = 'CSA07_perfect_L3Absolute_fastjet4'
string label = 'L3AbsoluteJetCorrectorFKt4'
}
//*************** Modules ***************************************
module L3JetCorJetIcone5 = JetCorrectionProducer {
InputTag src    = L2JetCorJetIcone5
vstring correctors = {"L3AbsoluteJetCorrectorIcone5"}
untracked string alias = "L3JetCorJetIcone5"
}
```

Important Files:

JetMETCorrections/Objetcs
- JetCorrector (BaseClass)

JetMETCorrections/Modules
- JetCorrection
- JetCorrectionService

[1)] There is an option to do this on the fly (w/o persistent collections?)

Software Structure

Hamburg

- Jet correction based on simple text files
- Option of DB access foreseen
- Single *scale factor* on reco::CaloJet::p4()
- Correction off CaloTowers not foreseen

Line: η_{min} | η_{max} | N_{par} | <par>

parse txt files → deduce corrections as function of η/p_t → Interface jets to corrections

Important Classes:

CondFormats/JetMETObjetcs
- *CorrectorParameters
- Simple*Corrector

JetMETCorerections/Algorithms
- *Corrector

Parsing of Calibration Constants

Hamburg

GlobalFitCorrectorParameters:

Line: η_{bin} | ϕ_{bin} | N_{twr} | N_{jet} |<twr> | <jet>

```
class GlobalFitCorrectorParameters{

 public:

  // keeps different parametrizations for the
  // underlying hypothesis of the GlobalFit
  class Parametrization{

  public:

    Parametrization(unsigned t, unsigned j) :
      nTowerParams_(t), nJetParams_(j) {}
    virtual ~Parametrization(){}
    // ------------------------------------------------------------------
    // correctedTowerEt(double *x, double *par)
    // returns the corrected Et of a tower
    // input: x[4]: x[0] : Et of whole tower
    //              x[1] : Et of ECAL part
    //              x[2] : Et of HCAL part
    //              x[3] : Et of Outer part
    // par:  the correction parameters of this tower
    // ------------------------------------------------------------------
    virtual double correctedTowerEt(double *x, double *par) const = 0;

    // ------------------------------------------------------------------
    // correctedJetEt(double *x, double *par)
    // returns the corrected Et of a jet
    // input: x[3]: x[0] : Et of uncorrected jet
    //              x[1] : eta of uncorrected jet(not used)
    //              x[2] : phi of uncorrected jet(not used)
    // par:  the correction parameters of this jet
    // ------------------------------------------------------------------
    virtual double correctedJetEt(double *x, double *par) const = 0;
    virtual const char* name() const = 0;

    unsigned nTowerPars() const { return nTowerParams_;}
    unsigned nJetPars() const { return nJetParams_;}

  private:

    Parametrization();
    unsigned nTowerParams_;
    unsigned nJetParams_;
  };

  // parametrization of the hadronic response
  // by a step function
  class StepParametrization : public Parametrization{

  public:

    StepParametrization() : Parametrization(12,2){}
```

```
// keeps eta/phi indices and the parameters
// for towers and jets
class Record{

public:
  Record():iEta_(0),iPhi_(0) {}
  Record(int eta, int phi, const std::vector<float>& tower, const std::vector<float>& jet) :
    iEta_(eta), iPhi_(phi), jetParameters_(jet), towerParameters_(tower) {}
  Record(const std::string&);
  int iEta() const {return iEta_;}
  int iPhi() const {return iPhi_;}
  unsigned nTowerParameters() const {return towerParameters_.size();}
  std::vector<float> towerParameters() const {return towerParameters_;}
  float towerParameter(unsigned idx) const {return towerParameters_[idx];}
  unsigned nJetParameters() const {return jetParameters_.size();}
  std::vector<float> jetParameters() const {return jetParameters_;}
  float jetParameter(unsigned idx) const {return jetParameters_[idx];}
  int operator< (const Record& other) const {return (iEta()<other.iEta() ? iPhi()<other.iPhi() : false);}

private:
  int iEta_;
  int iPhi_;
  std::vector<float> jetParameters_;
  std::vector<float> towerParameters_;
};
```

```
GlobalFitCorrectorParameters(){}
GlobalFitCorrectorParameters(const std::string&, const std::string& section = "");
~GlobalFitCorrectorParameters(){ delete parametrization_; }

/// total # of eta phi bins
unsigned size() const {return records_.size();}
/// total # of eta bins
unsigned etaSize() const {return etaSize_;}
/// total # of phi bins
unsigned phiSize() const {return ((records_.size()-1)%etaSize_+1);}
/// get param index for eta and phi
unsigned parIndex(float, float) const;
/// get record for the band
const Record& record(unsigned par) const {return records_[par];}
/// get record for the band
const Record& record(int ieta, int iphi) const {return records_[parIndex(ieta, iphi)]; };
/// get parametrization
const Parametrization& parametrization() {return *parametrization_; };

private:
 unsigned etaSize_;
 std::vector<Record> records_;
 Parametrization* parametrization_;
```

Application of Parameters

Hamburg

GlobalFitCorrector:

```
#ifndef GlobalFitCorrector_h
#define GlobalFitCorrector_h

#include <map>
#include <string>
#include <vector>

namespace reco{
  class CaloJet;
}

class GlobalFitCorrectorParameters;

class GlobalFitCorrector{
 public:
  GlobalFitCorrector() : params_(0) {};
  GlobalFitCorrector(const std::string&);
  virtual ~GlobalFitCorrector();

  /// apply correction using CaloJet information
  virtual double correction(const reco::CaloJet&) const;

 private:
  GlobalFitCorrector(const GlobalFitCorrector&);
  GlobalFitCorrector& operator=(const GlobalFitCorrector&);

 private:
  typedef std::pair<int,int>  CalibKey;
  typedef std::vector<double> CalibVal;
  typedef std::map<CalibKey,CalibVal> CalibMap;

  GlobalFitCorrectorParameters* params_;
};

#endif
```

```
double GlobalFitCorrector::correction(const reco::CaloJet& jet) const
{
  double correction=1.;
  reco::Particle::LorentzVector towerCorrected;

  // ------------------------------------------
  // prepare calibration maps
  // ------------------------------------------
  CalibMap towerParams, jetParams;
  for(unsigned int idx=0; idx<params_->size(); ++idx){
    for(unsigned int par=0; par<params_->record(idx).towerParameters().size(); ++par)
      towerParams[CalibKey(params_->record(idx).iEta(),params_->record(idx).iPhi())].push_back(params_->
par));
    for(unsigned int par=0; par<params_->record(idx).jetParameters().size(); ++par)
      jetParams  [CalibKey(params_->record(idx).iEta(),params_->record(idx).iPhi())].push_back(params_->
r)  );
  }

  // ------------------------------------------
  // apply calo tower correction and
  // sum tower corrected constituents
  // ------------------------------------------
  std::vector<double> towerDeps;
  for(std::vector <CaloTowerPtr>::const_iterator tower = jet.getCaloConstituents().begin() ;
      tower!=jet.getCaloConstituents().end(); ++tower) {
    const CaloTowerDetId& id = (*tower)->id();

    float scale = 1.;
    if( (*tower)->et()>0. ){
      // prepare dependencies
      towerDeps.push_back( (*tower)->et()      );
      towerDeps.push_back( (*tower)->emEt()    );
      towerDeps.push_back( (*tower)->hadEt()   );
      towerDeps.push_back( (*tower)->outerEt() );

      // read tower parameters back
      // and apply tower calibration
      CalibMap::const_iterator pars = towerParams.find(CalibKey(id.ieta(),id.iphi()));
      if( pars != towerParams.end() ){
        CalibVal val=pars->second;
        scale = params_->parametrization().correctedTowerEt(&towerDeps[0], &val[0])/(*tower)->et();
      }
    }
    // create calibrated tower
    towerCorrected += scale*(*tower)->p4();
  }

  // ------------------------------------------
  // find closest tower to jet in DeltaR
  // for subsequent jet correction
  // ------------------------------------------
  int jetIdx = 0;
  double dist=-1;
  int closestTower=-1;
  for(std::vector <CaloTowerPtr>::const_iterator tower = jet.getCaloConstituents().begin() ;
      tower!=jet.getCaloConstituents().end(); ++tower, ++jetIdx) {
    double dR=deltaR(towerCorrected.eta(), towerCorrected.phi(), (*tower)->eta(), (*tower)->phi());
    if( dist<0 || dR<dist ){
      dist = dR;
      closestTower = jetIdx;
    }
  }

  // ------------------------------------------
  // apply subsequent jet correction
  //
```

Wrapper of Correction

Hamburg

GlobalFitJetCorrector:

```
class GlobalFitJetCorrector : public JetCorrector{
 public:
  GlobalFitJetCorrector(const edm::ParameterSet&);
  virtual ~GlobalFitJetCorrector();

  /// apply correction using Jet information only
  virtual double correction (const reco::Jet&) const;

  /// if correction needs event information
   virtual bool eventRequired () const {return false;}

 private:
  GlobalFitCorrector* corrector_;
};
```

```
GlobalFitJetCorrector::GlobalFitJetCorrector(const edm::ParameterSet& cfg)
{
 std::string file="CondFormats/JetMETObjects/data/"+cfg.getParameter<std::string>("tagName")+".txt";
 edm::FileInPath f1(file);
 corrector_ = new GlobalFitCorrector( f1.fullPath() );
}

GlobalFitJetCorrector::~GlobalFitJetCorrector()
{
  delete corrector_;
}

double GlobalFitJetCorrector::correction(const reco::Jet& jet) const
{
  const reco::CaloJet* caloJet = dynamic_cast<const reco::CaloJet*>(&jet);
  if( caloJet!=0 )
    return corrector_->correction(*caloJet);
  else
    return 1.;
}
```

ToDo`s

- Stream out correct file format from GlobalFit
- Restructure as far as appropriate (→)
- Test everything during next calibration round (in 210 only)

Betreff: Re: added JetCorrector for GlobalFit ansatz to CondFormats/JetMETObjects and JetMETCorrections/Algorithms

Von: Fedor Ratnikov <ratnikov@fnal.gov>

Datum: Tue, 1 Jul 2008 15:21:15 -0500

An: Roger Wolf <roger.wolf@cern.ch>

CC: <rharris@fnal.gov>, <Roger.Wolf@desy.de>

Hi Roger,
I guess I never answered this, sorry.

Before we put this new option into production, I'd like to consider
few cosmetics improvements.

- GlobalFitCorrectorParameters is now essentially responsible for parametrization,
 and GlobalFitCorrector is a wrapper around it.
It would be extremely useful if you could re-use SimpleJetCorrectorParameters
to interface parameters to parameter files. This would significantly simplify
future transition to storing jet correction parameters in the DB.
Also our practice is that corresponding xxxCorrector is responsible for
interpreting parameters and converting generic SimpleJetCorrectorParameters
to concrete correction.

- GlobalFitCorrector::correction function starts with unwinding parameters
 into jetParams and towerParams.
Could this be done once after parameters are available, and be re-used then?

Thank you!
-F