{{lastupdated_at}} by {{lastupdated_by}}

TRA file format

The TRA file format is used by MEGALib to store event information. Check the MEGALib methods src/revan/src/MRERawEvent::ParseLine to better understand how to interpret the format. Note the inheritance of this class:

```
class MRERawEvent : public MRESE, public MRotationInterface
```

TI

Parameters:

1. [float] Defines the event time in seconds. Time zero corresponds to 1970-01-01T01:00:00.

Here is the relevant code in src/revan/src/MRERawEvent::ParseLine:

```
m_EventTime.Set(Line)
bool MTime::Set(const char* Line)
m_Seconds
m_NanoSeconds
}
```

Here a GammaLib code snippet for TI=1465776080.278382500 that gives the event time in UTC:

```
time  = gammalib.GTime('1970-01-01T01:00:00')
time += 1465776080.278382500
print(time.utc())
2016-06-13T01:00:54
```

RX

Parameters:

1. [float] m_DetectorRotationXAxis[0] [cm] (default: 1.0)
2. [float] m_DetectorRotationXAxis[1] [cm] (default: 0.0)
3. [float] m_DetectorRotationXAxis[2] [cm] (default: 0.0)

RZ

Parameters:

1. [float] m_DetectorRotationZAxis[0] [cm] (default: 0.0)
2. [float] m_DetectorRotationZAxis[1] [cm] (default: 0.0)
3. [float] m_DetectorRotationZAxis[2] [cm] (default: 1.0)

GX

Galactic longitude of detector X axis.
Parameters:

1. [float] Longitude [deg]
2. [float] Latitude [deg]

```
SetGalacticPointingXAxis(Longitude, Latitude)
```

GZ

Galactic longitude of detector Z axis.
Parameters:

1. [float] Longitude [deg]
2. [float] Latitude [deg]

```
SetGalacticPointingZAxis(Longitude, Latitude)
```

HX

Horizon of detector X axis.
Parameters:

1. [float] Azimuth [deg]
2. [float] Altitude [deg]

The information is set by

```
m_HorizonPointingXAxis.SetMagThetaPhi(1.0, (90-Altitude)*c_Rad, Azimuth*c_Rad)
```

which uses the MVector::SetMagThetaPhi method

```
void MVector::SetMagThetaPhi(double Magnitude, double Theta, double Phi)
{
  // Set in spherical coordinates

  double A = fabs(Magnitude);
  m_X = A*sin(Theta)*cos(Phi);
  m_Y = A*sin(Theta)*sin(Phi);
  m_Z = A*cos(Theta);
}
```

HZ

Horizon of detector Z axis.
Parameters:

1. [float] Azimuth [deg]
2. [float] Altitude [deg]

The same method

```
m_HorizonPointingZAxis.SetMagThetaPhi(1.0, (90-Altitude)*c_Rad, Azimuth*c_Rad)
```

is used (see HX)

CE

Energy of Compton event, sets members of the MComptonEvent class.
Parameters:

1. [float] m_Eg [keV]
2. [float] m_dEg [keV]
3. [float] m_Ee [keV]
4. [float] m_dEe [keV]

CD

Direction of Compton events, sets members of the MComptonEvent class.
Parameters:

1. [float] m_C1[0] [cm]
2. [float] m_C1[1] [cm]
3. [float] m_C1[2] [dm]
4. [float] m_dC1[0] [cm]
5. [float] m_dC1[1] [cm]
6. [float] m_dC1[2] [dm]
7. [float] m_C2[0] [cm]
8. [float] m_C2[1] [cm]
9. [float] m_C2[2] [dm]
10. [float] m_dC2[0] [cm]
11. [float] m_dC2[1] [cm]
12. [float] m_dC2[2] [dm]
13. [float] m_Ce[0] [cm]
14. [float] m_Ce[1] [cm]
15. [float] m_Ce[2] [dm]
16. [float] m_dCe[0] [cm]
17. [float] m_dCe[1] [cm]
18. [float] m_dCe[2] [dm]

TL

Track length, sets member of MComptonEvent class.
Parameters:

1. [float] m_TrackLength [cm]

TE

Track initial deposit, sets member of MComptonEvent class.
Parameters:

1. [float] m_TrackInitialDeposit [keV]

LA

Lever arm, sets member of MComptonEvent class.
Parameters:

1. [float] m_LeverArm [cm]

SQ

Sequence length, sets member of MComptonEvent class.
Parameters:

1. [int] m_SequenceLength

PQ

Clustering quality factor, sets member of MComptonEvent class.
Parameters:

1. [float] m_ClusteringQualityFactor

CT

Compton quality factor, sets member of MComptonEvent class.
Parameters:

1. [float] m_ComptonQualityFactor1
2. [float] m_ComptonQualityFactor2 (optional, default: 1.0)

Computation of Chi and Psi

In the detector system, the event scatter direction is defined by Chi and Psi, where Chi is an azimuth angle and Psi is a zenith angle. The computation of Chi and Psi is done in MEGAlib in src/response/src/MResponseImagingBinnedMode::Analyze:

```
// Get the data space information
MRotation Rotation = Compton->GetDetectorRotationMatrix();

double Phi = Compton->Phi()*c_Deg;
MVector Dg = -Compton->Dg();
Dg = Rotation*Dg;
double Chi = Dg.Phi()*c_Deg;
while (Chi < -180) Chi += 360.0;
while (Chi > +180) Chi -= 360.0;
double Psi = Dg.Theta()*c_Deg;
```

where

```
// Direction of the second gamma-ray:
m_Dg = (m_C2 - m_C1).Unit();
```

is the direction of the second gamma-ray, defined in src/misc/src/MComptonEvent::Validate.

Computation of Phi

The Compton scatter angle is computed in MEGAlib in src/global/misc/src/MComptonEvent::CalculatePhi

```
bool MComptonEvent::CalculatePhi()
{
  // Compute the compton scatter angle due to the standard equation
  // i.e neglect the movement of the electron,
  // which would lead to a Doppler-broadening
  //
  // Attention, make sure to test before you call this method,
  //            that the Compton-kinematics is correct

  double Value = 1 - c_E0 * (1/m_Eg - 1/(m_Ee + m_Eg));

  if (Value <= -1 || Value >= 1) {
    return false;
  }

  m_Phi = acos(Value);

  return true;
}
```

Computation of the Earth horizon angle

In MEGAlib, an intersection test with the Earth horizon is done in
src/mimrec/src/MEarthHorizon::IsEventFromEarthByIntersectionTest

```
bool MEarthHorizon::IsEventFromEarthByIntersectionTest(MPhysicalEvent* Event, bool DumpOutput) const
{
  if (Event->GetType() == MPhysicalEvent::c_Compton) {
    MComptonEvent* C = dynamic_cast<MComptonEvent*>(Event);

    double Phi = C->Phi();

    MVector ConeAxis = -C->Dg();

    // Rotate the ConeAxis into the Earth system:
    ConeAxis = C->GetHorizonPointingRotationMatrix()*ConeAxis;

    // Distance between the Earth cone axis and the Compton cone axis:
    double AxisDist = m_PositionEarth.Angle(ConeAxis);

    if (fabs(AxisDist - Phi) > m_HorizonAngle) {
      return false;
    } else {
      if (DumpOutput == true) {
        mout<<"ID "<<Event->GetId()<<": Cone intersects Earth: "<<AxisDist + Phi<<" < "<<m_HorizonAngle<<endl;
      }
      return true;
    }
  } else if (Event->GetType() == MPhysicalEvent::c_Pair) {
    MPairEvent* P = dynamic_cast<MPairEvent*>(Event);
    double AxisDist = m_PositionEarth.Angle(P->GetOrigin());

    if (AxisDist < m_HorizonAngle) {
      if (DumpOutput == true) {
        mout<<"ID "<<Event->GetId()<<": Origin inside Earth: "<<AxisDist<<" < "<<m_HorizonAngle<<endl;
      }
      return true;
    }
  }

  return false;
}
```

Note that

```
MRotation MRotationInterface::GetHorizonPointingRotationMatrix() const
{
  // Return the rotation matrix of this event

  // Verify that x and z axis are at right angle:
  if (fabs(m_HorizonPointingXAxis.Angle(m_HorizonPointingZAxis) - c_Pi/2.0)*c_Deg > 0.1) {
    cout<<"Event "<<m_Id<<": Horizon axes are not at right angle, but: "
<<m_HorizonPointingXAxis.Angle(m_HorizonPointingZAxis)*c_Deg<<" deg"<<endl;
  }

  // First compute the y-Axis vector:
  MVector m_HorizonPointingYAxis = m_HorizonPointingZAxis.Cross(m_HorizonPointingXAxis);

  return MRotation(m_HorizonPointingXAxis.X(), m_HorizonPointingYAxis.X(), m_HorizonPointingZAxis.X(),
                   m_HorizonPointingXAxis.Y(), m_HorizonPointingYAxis.Y(), m_HorizonPointingZAxis.Y(),
                   m_HorizonPointingXAxis.Z(), m_HorizonPointingYAxis.Z(), m_HorizonPointingZAxis.Z());
}
```

and

```
double MVector::Angle(const MVector& V) const
{
  // Calculate the angle between two vectors:
  // cos Angle = (v dot w) / (|v| x |w|)

  // Protect against division by zero:
  double Denom = Mag()*V.Mag();
  if (Denom == 0) {
    return 0.0;
  } else {
    double Value = Dot(V)/Denom;
    if (Value >  1.0) Value =  1.0;
    if (Value < -1.0) Value = -1.0;
    return acos(Value);
  }
}
```

and

```
m_PositionEarth = MVector(0, 0, -1);
```

which is the position of Earth (center) in detector coordinates and

```
m_HorizonAngle = 90*c_Rad;
```

is the (azimuth-) angle from the Earth position to Earth horizon (the values indicate the default values of the MEarthHorizon constructor). The values are set using

```
bool MEarthHorizon::SetEarthHorizon(const MVector& PositionEarth,
                                    const double HorizonAngle)
{
  m_PositionEarth = PositionEarth.Unit();
  m_HorizonAngle = HorizonAngle;

  return true;
}
```

which is called in src/mimrec/src/MEventSelector.cxx.

Alternatively, a probabilistic evaluation is done using

```
bool MEarthHorizon::IsEventFromEarthByProbabilityTest(MPhysicalEvent* Event, bool DumpOutput) const
{
  massert(Event != 0);

  if (Event->GetType() == MPhysicalEvent::c_Compton) {
    MComptonEvent* C = dynamic_cast<MComptonEvent*>(Event);

    // Take care of scatter angles larger than 90 deg:
    double Phi = C->Phi();
    MVector ConeAxis = C->Dg();
    // Rotate the ConeAxis into the Earth system:
    ConeAxis = C->GetHorizonPointingRotationMatrix()*ConeAxis;

    MVector Origin = C->DiOnCone();

    // That's the trick, but I don't remember what it means...
    if (Phi > c_Pi/2.0) {
      Phi = c_Pi - Phi;
```

```
  } else {
    ConeAxis *= -1;
  }

  // Distance between the Earth cone axis and the Compton cone axis:
  double EarthConeaxisDist = m_PositionEarth.Angle(ConeAxis);

  // Now determine both angles between the cone and Earth (simpler now since phi always <= 90)
  // First the one towards the Earth -- if it is smaller than 0 just use fabs
  double AngleEarthCone1 = EarthConeaxisDist - Phi;
  if (AngleEarthCone1 < 0) AngleEarthCone1 = fabs(AngleEarthCone1); // please don't simplify
  // Then the one away from Earth -- if it is > 180 deg then use the smaller angle
  double AngleEarthCone2 = EarthConeaxisDist + Phi;
  if (AngleEarthCone2 > c_Pi) AngleEarthCone2 -= 2*(AngleEarthCone2 - c_Pi);

  // Case a: Cone is completely inside Earth
  if (AngleEarthCone1 < m_HorizonAngle && AngleEarthCone2 < m_HorizonAngle) {
    mdebug<<"EHC: Cone is completely inside Earth"<<endl;
    if (m_MaxProbability < 1.0) {
      if (DumpOutput == true) {
        mout<<"ID "<<Event->GetId()<<": Cone inside Earth"<<endl;
      }
      return true;
    } else {
      // If the probability is 1.0, we are OK with events from Earth
      return false;
    }
  }
  // Case b: Cone is completely outside Earth
  else if (AngleEarthCone1 > m_HorizonAngle && AngleEarthCone2 > m_HorizonAngle) {
    mdebug<<"EHC: Cone is completely outside Earth"<<endl;
    return false;
  }
  // Case c: Cone intersects horizon
  else {
    // Determine the intersection points on the Compton cone:
    mdebug<<"EHC: Cone intersects horizon"<<endl;

    if (sin(EarthConeaxisDist) == 0 || sin(Phi) == 0) {
      merr<<"Numerical boundary: Scattered gamma-ray flew in direction of "
          <<"the Earth axis and the Compton scatter angle is identical with "
          <<"horizon angle (horizon angle="<<m_HorizonAngle*c_Deg
          <<", Earth-coneaxis-distance="<<EarthConeaxisDist*c_Deg
          <<") - or we have a Compton backscattering (180 deg) "
          <<"or no scattering at all (phi="<<Phi*c_Deg<<")... "
          <<"Nevertheless, I am not rejecting event "<<Event->GetId()<<show;
      return false;
    }
    // Law of cosines for the sides of spherical triangles
    double EarthConeaxisIntersectionAngle =
      acos((cos(m_HorizonAngle)-cos(EarthConeaxisDist)*cos(Phi))/(sin(EarthConeaxisDist)*sin(Phi)));

    if (C->HasTrack() == true && m_ValidComptonResponse == true) {
      // Now we have to determine the distance to the origin on the cone:

      double ConeaxisOriginDist = ConeAxis.Angle(Origin);
      double EarthOriginDist = m_PositionEarth.Angle(Origin);

      if (sin(ConeaxisOriginDist) == 0) {
        merr<<"Numerical boundary: The photon's origin is identical with the cone axis...!"
            <<" Nevertheless, I am not rejecting this event "<<Event->GetId()<<show;
        return false;
      }

      // Law of cosines for the sides of spherical triangles
      double EarthConeaxisOriginAngle =
        acos((cos(EarthOriginDist) - cos(EarthConeaxisDist)*cos(ConeaxisOriginDist))/
```

```
            (sin(EarthConeaxisDist)*sin(ConeaxisOriginDist)));

      // The intersections on the cone are at EarthConeaxisOriginAngle +-EarthConeaxisIntersectionAngle
      // Let's figure out the probabilities:

      double Probability = 0;

      // Case A: "Maximum" on cone (not necessarily origin) is from *within* Earth:
      if (EarthOriginDist < m_HorizonAngle) {
        // If the intersection are in different Origin-Coneaxis-Hemispheres:
        if (EarthConeaxisOriginAngle + EarthConeaxisIntersectionAngle < c_Pi) {
          Probability = m_ComptonResponse.GetInterpolated((EarthConeaxisIntersectionAngle - EarthConeaxisOriginAngle)*c_Deg,
C->Ee()) +
            m_ComptonResponse.GetInterpolated((EarthConeaxisIntersectionAngle + EarthConeaxisOriginAngle)*c_Deg, C->Ee());
          Probability = 0.5*Probability;

          massert(EarthConeaxisIntersectionAngle - EarthConeaxisOriginAngle >= 0);
          massert(EarthConeaxisIntersectionAngle - EarthConeaxisOriginAngle <= c_Pi);
          massert(EarthConeaxisIntersectionAngle + EarthConeaxisOriginAngle >= 0);
          massert(EarthConeaxisIntersectionAngle + EarthConeaxisOriginAngle <= c_Pi);

        }
        // otherwise:
        else {
          Probability = m_ComptonResponse.GetInterpolated((2*c_Pi - EarthConeaxisIntersectionAngle -
EarthConeaxisOriginAngle)*c_Deg, C->Ee()) -
            m_ComptonResponse.GetInterpolated((EarthConeaxisIntersectionAngle - EarthConeaxisOriginAngle)*c_Deg, C->Ee());
          Probability = 1-0.5*Probability;
        }
      }
      // Case B: "Maximum" on cone (not necessarily origin) is from *outside* Earth:
      else {
        // If the intersection are in different hemispheres Origin-Coneaxis-Hemispheres:
        if (EarthConeaxisOriginAngle + EarthConeaxisIntersectionAngle > c_Pi) {
          Probability = m_ComptonResponse.GetInterpolated((EarthConeaxisOriginAngle - EarthConeaxisIntersectionAngle)*c_Deg,
C->Ee()) +
            m_ComptonResponse.GetInterpolated((c_Pi + EarthConeaxisIntersectionAngle - EarthConeaxisOriginAngle)*c_Deg,
C->Ee());
          Probability = 1-0.5*Probability;
        }
        // otherwise:
        else {
          Probability = m_ComptonResponse.GetInterpolated((EarthConeaxisOriginAngle + EarthConeaxisIntersectionAngle)*c_Deg,
C->Ee()) -
            m_ComptonResponse.GetInterpolated((EarthConeaxisOriginAngle - EarthConeaxisIntersectionAngle)*c_Deg, C->Ee());
          Probability = 0.5*Probability;

        }
      }

      if (Probability > m_MaxProbability) {
        if (DumpOutput == true) {
          mout<<"ID "<<Event->GetId()<<": Probability higher max probability: "<<Probability<<" > "<<m_MaxProbability<<endl;
        }
        return true;
      }
    } else {
      // The probability is simply determined by the length of the segment within Earth
      if (EarthConeaxisIntersectionAngle/c_Pi > m_MaxProbability) {
        if (DumpOutput == true) {
          mout<<"ID "<<Event->GetId()<<": Probability higher max probability: "<<EarthConeaxisIntersectionAngle/c_Pi<<" > "
<<m_MaxProbability<<endl;
        }
        return true;
      }
    }
  }
```

```
  } else if (Event->GetType() == MPhysicalEvent::c_Pair) {
    MPairEvent* P = dynamic_cast<MPairEvent*>(Event);
    double AxisDist = m_PositionEarth.Angle(P->GetOrigin());

    if (AxisDist < m_HorizonAngle) {
      if (DumpOutput == true) {
        mout<<"ID "<<Event->GetId()<<": Origin inside Earth: "<<AxisDist<<" < "<<m_HorizonAngle<<endl;
      }
      return true;
    }
  }

  return false;
}
```

MEGAlib constants

The following constants are used in MEGAlib:

```
const double c_Pi = 3.14159265358979311600;
const double c_TwoPi = 2*c_Pi;
const double c_Sqrt2Pi = 2.506628274631;
const double c_Rad = c_Pi / 180.0;
const double c_Deg = 180.0 / c_Pi;
const double c_SpeedOfLight = 29.9792458E+9; // cm/s
const double c_E0 = 510.999; // keV
const double c_FarAway = 1E20; // cm
const double c_LargestEnergy = 0.999*numeric_limits<float>::max();
const MVector c_NullVector(0.0, 0.0, 0.0);
```