TTree Example

TTree Example

This example is taken from the SVN repository: ./Examples/TTree/Test.java.

It creates a ROOT file (FromJava.root), and creates a TTree called "demoT", with branches:

- Floats: fX, fY, fZ.
- Ints: iX, iY, iZ.
- TVector3: vec1
- std::vector<float>: vecF.
- std::vector<int>: vecI.
- std::vector<double>: vecD.
- std::vector<bool>: vecB.
- std::vector<TVector3>: vec3.

Three points are added to each of the vectors of primitive types, and a random number of points in [1,16] is added to the vector<TVector3>.

The second TTree has branches:

- Doubles: dX, dY, dZ.
- Bool: bX, bY, bZ.
- TLorentzVector: lvec1.
- std::vector<TLorentzVector>: vecL.

```
import java.util.Random;
import hep.physics.vec.*;
import org.lcsim.thirdparty.javaROOT.*;

public class Test
{
        public static void main(String[] args)
        {
                // Load the library
                try
                {
                        System.loadLibrary( "javaROOT" );
                }
                catch( UnsatisfiedLinkError javaROOT_linkError )
                {
                        System.err.println( "JavaROOT native library failed to load.\n" + javaROOT_linkError +
"\n" );
                        System.exit( -1 );
                }

                // Create a RNG to fill our histograms with.
                Random r = new Random( 0 );

                // Create a file
                RootSessionField sess = new RootSessionField( "FromJava.root", "RECREATE", "Test", 1 );

                // Create a tree, populate it with branches and set it up for use.
                sess.newTTree( "demoT", "TestTree", 99 );

                // The branch names are initially cached. We cannot fill the tree with
                // data until we "setup" the TTree with RootSessionField::setupTTree,
                // when the branches are actually created in the ROOT file.
                // At this point, no further branches can be created, but we can fill
                // tree with data.

                // Branches of type float, called "fX", "fY" and "fZ" respectively.
                sess.branchTTreeFloat( "demoT", "fX" );
                sess.branchTTreeFloat( "demoT", "fY" );
                sess.branchTTreeFloat( "demoT", "fZ" );
                // Branches of type int, called "iX", etc.
                sess.branchTTreeInt( "demoT", "iX" );
                sess.branchTTreeInt( "demoT", "iY" );
                sess.branchTTreeInt( "demoT", "iZ" );
                // Branch of type TVector3.
```

```
                sess.branchTTreeVector3( "demoT", "vec1" );
                // Branch of type std::vector<float>.
                sess.branchTTreeVectorFloat( "demoT", "vecF" );
                // Branch of type std::vector<int>.
                sess.branchTTreeVectorInt( "demoT", "vecI" );
                // Branch of type std::vector<double>.
                sess.branchTTreeVectorDouble( "demoT", "vecD" );
                // Branch of type std::vector<bool>.
                sess.branchTTreeVectorBool( "demoT", "vecB" );
                // Branch of type std::vector<TVector3>.
                sess.branchTTreeVectorVector3( "demoT", "vec3" );

                // Setup the tree to fill with data.
                sess.setupTTree( "demoT" );

                // Create another tree
                sess.newTTree( "demoU", "AnotherTestTree", 99 );
                // Branches of type double.
                sess.branchTTreeDouble( "demoU", "dX" );
                sess.branchTTreeDouble( "demoU", "dY" );
                sess.branchTTreeDouble( "demoU", "dZ" );
                // Branches of type bool.
                sess.branchTTreeBool( "demoU", "bX" );
                sess.branchTTreeBool( "demoU", "bY" );
                sess.branchTTreeBool( "demoU", "bZ" );
                // Branch of type TLorentzVector.
                sess.branchTTreeLorentzVector( "demoU", "lvec1" );
                // Branch of type std::vector<TLorentzVector>.
                sess.branchTTreeVectorLorentzVector( "demoU", "vecL" );

                // Setup the tree to fill with data.
                sess.setupTTree( "demoU" );

                // Currently, you must use these write-only collection classes to
                // pass vectors to javaROOT. These are included in
                // org.lcsim.thirdparty.javaROOT.
                vectorFloat fVals;
                vectorInt iVals;
                vectorDouble dVals;
                vectorBool bVals;
                vectorVector3 vecVals;
                vectorLorentzVector lorVals;

                for( int i = 0; i < 4096; i++ )
                {
                        // (Re-)create the collections, clearing them.
                        fVals = new vectorFloat();
                        iVals = new vectorInt();
                        dVals = new vectorDouble();
                        bVals = new vectorBool();
                        vecVals = new vectorVector3();
                        lorVals = new vectorLorentzVector();

                        sess.fillBranchFloat( "demoT", "fZ", (float) r.nextGaussian() );
                        sess.fillBranchFloat( "demoT", "fX", (float) r.nextGaussian() );
                        sess.fillBranchFloat( "demoT", "fY", (float) r.nextGaussian() );

                        sess.fillBranchInt( "demoT", "iZ", (int) ( r.nextGaussian() * 1024 ) );
                        sess.fillBranchInt( "demoT", "iY", (int) ( r.nextGaussian() * 1024 ) );
                        sess.fillBranchInt( "demoT", "iX", (int) ( r.nextGaussian() * 1024 ) );

                        // For convenience, we pass hep.physics.vec.Hep3Vectors to fillBranchVector3,
                        // which is automatically stored as a ROOT TVector3 in the ROOT file.
                        BasicHep3Vector t = new BasicHep3Vector( r.nextGaussian(), r.nextGaussian(), r.
nextGaussian() );
                        sess.fillBranchVector3( "demoT", "vec1", t );

                        // Generate some data, and add them to the collections.
                        fVals.add( (float) r.nextGaussian() );
                        fVals.add( (float) r.nextGaussian() );
                        fVals.add( (float) r.nextGaussian() );
```

```
                        sess.fillBranchVectorFloat( "demoT", "vecF", fVals );

                        iVals.add( (int) ( r.nextGaussian() * 1024 ) );
                        iVals.add( (int) ( r.nextGaussian() * 1024 ) );
                        iVals.add( (int) ( r.nextGaussian() * 1024 ) );
                        sess.fillBranchVectorInt( "demoT", "vecI", iVals );

                        dVals.add( r.nextGaussian() );
                        dVals.add( 2 * r.nextGaussian() );
                        dVals.add( 3 * r.nextGaussian() + 2 );
                        sess.fillBranchVectorDouble( "demoT", "vecD", dVals );

                        bVals.add( ( r.nextGaussian() > 0 ) ? true : false );
                        bVals.add( ( r.nextGaussian() > 0 ) ? true : false );
                        bVals.add( ( r.nextGaussian() > 0 ) ? true : false );
                        sess.fillBranchVectorBool( "demoT", "vecB", bVals );

                        for( int j = 0; j < r.nextInt( 15 ) + 1; j++ )
                        {
                                vecVals.add( new BasicHep3Vector( (float) r.nextGaussian(), (float) r.
nextGaussian(), (float)
                                r.nextGaussian() ) );
                        }
                        sess.fillBranchVectorVector3( "demoT", "vec3", vecVals );

                        // "Commit" these data to the first tree.
                        sess.fillTTree( "demoT" );

                        // Fill the second tree with data.
                        sess.fillBranchDouble( "demoU", "dY", r.nextGaussian() );
                        sess.fillBranchDouble( "demoU", "dZ", 2 * r.nextGaussian() );
                        sess.fillBranchDouble( "demoU", "dX", 3 * r.nextGaussian() + 2 );

                        sess.fillBranchBool( "demoU", "bY", ( r.nextGaussian() > 0 ) ? true : false );
                        sess.fillBranchBool( "demoU", "bX", ( r.nextGaussian() > 0 ) ? true : false );
                        sess.fillBranchBool( "demoU", "bZ", ( r.nextGaussian() > 0 ) ? true : false );

                        // For convenience, we pass hep.physics.vec.HepLorentzVectors to
fillBranchLorentzVector,
                        // which is automatically stored as a ROOT TLorentzVector in the ROOT file.
                        BasicHepLorentzVector t2 = new BasicHepLorentzVector( r.nextGaussian(), r.
nextGaussian(), r.nextGaussian(), r.nextGaussian() );
                        sess.fillBranchLorentzVector( "demoU", "lvec1", t2 );

                        for( int j = 0; j < r.nextInt( 15 ) + 1; j++ )
                        {
                                lorVals.add( new BasicHepLorentzVector( (float) r.nextGaussian(), (float) r.
nextGaussian(), (float) r.nextGaussian(), (float)
                                r.nextGaussian() ) );
                        }
                        sess.fillBranchVectorLorentzVector( "demoU", "vecL", lorVals );

                        // Commit the data to the second tree.
                        sess.fillTTree( "demoU" );
                }

                // Essential to ensure all values are properly committed to the file.
                sess.delete();
        }
}
```