CMSSW/ DataFormats/ EcalDetId/ interface/ EEDetId.h

```
001 #ifndef ECALDETID_EEDETID_H
002 #define ECALDETID_EEDETID_H
003
004 #include <ostream>
005 #include "DataFormats/DetId/interface/DetId.h"
006 #include "DataFormats/EcalDetId/interface/EcalScDetId.h"
007 #include "DataFormats/EcalDetId/interface/EcalSubdetector.h"
008
009
010 /** \class EEDetId
011  *  Crystal/cell identifier class for the ECAL endcap
012  *
013  *
014  *  $Id: EEDetId.h,v 1.25 2010/03/03 18:52:39 ferriff Exp $
015  */
016 class EEDetId : public DetId {
017    public:
018       enum {
019          /** Sudetector type. Here it is ECAL endcap.
020           */
021          Subdet=EcalEndcap
022       };
023
024       /** Constructor of a null id
025        */
026       EEDetId() {}
027
028       /** Constructor from a raw value
029        * @param rawid det ID number
030        */
031       EEDetId(uint32_t rawid) : DetId(rawid) {}
032
033       /** Constructor from crystal ix,iy,iz (iz=+1/-1) (mode = XYMODE)
034        * or from sc,cr,iz (mode = SCCRYSTALMODE).
035        * <p>ix runs from 1 to 100 along x-axis of standard CMS coordinates<br>
036        * iy runs from 1 to 100 along y-axis of standard CMS coordinates<br>
037        * iz is -1 for EE- and +1 for EE+<br>
038        * <p>For isc see isc(), for ic see ic()
039        * @see isc(), ic()
040        * @param i ix or isc index
041        * @param j iy or isc index
042        * @param iz iz/zside index: -1 for EE-, +1 for EE+
043        * @param mode pass XYMODE if i j refer to ix, iy, SCCRYSTALMODE if thery refer to isc,
044        */
045       EEDetId(int i, int j, int iz, int mode = XYMODE);
046
047       /** Constructor from a generic cell id
048        * @param id source detid
049        */
050       EEDetId(const DetId& id);
051
052       /** Assignment operator
053        * @param id source det id
054        */
055       EEDetId& operator=(const DetId& id);
056
057       /** Gets the subdetector
058        * @return subdetectot ID, that is EcalEndcap
059        */
060       static EcalSubdetector subdet() { return EcalEndcap;}
061
062       /** Gets the z-side of the crystal (1/-1)
063        * @return -1 for EE-, +1 for EE+
064        */
```

```
065       int zside() const { return (id_&0x4000)?(1):(-1); }
066
067       /** Gets the crystal x-index.
068        * @see EEDetId(int, int, int, int) for x-index definition
069        * @return x-index
070        */
071       int ix() const { return (id_>>7)&0x7F; }
072
073       /** Get the crystal y-index
074        * @see EEDetId(int, int, int, int) for y-index definition.
075        * @return y-index
076        */
077       int iy() const { return id_&0x7F; }
078
079       /** Gets the DetId of the supercrystal the crystal belong to.
080        * @return the supercrystal det id
081        * @throw cms::Exception if the crystal det id is invalid
082        */
083       EcalScDetId sc() const {
084          const int scEdge = 5;
085          return EcalScDetId(1+(ix()-1)/scEdge, 1+(iy()-1)/scEdge, zside());
086       }
087
088       /** Gets the SuperCrystal number within the endcap. This number runs from 1 to 316,
089        * numbers 70 149 228 307 are not used.
090        *
091        * BEWARE: This number is not consistent with indices used in constructor:  see details
092        *
093        * Numbering in quadrant 1 of EE+ is the following
094        * \verbatim
095        *  08 17 27
096        *  07 16 26 36 45 54
097        *  06 15 25 35 44 53 62
098        *  05 14 24 34 43 52 61 69
099        *  04 13 23 33 42 51 60 68 76
100        *  03 12 22 32 41 50 59 67 75
101        *  02 11 21 31 40 49 58 66 74
102        *  01 10 20 30 39 48 57 65 73 79
103        *     09 19 29 38 47 56 64 72 78
104        *        18 28 37 46 55 63 71 77
105        *
106        *        == THERE IS NO INDEX 70! ==
107        * \endverbatim
108        *
109        * Quadrant 2 indices are deduced by a symetry about y-axis and by adding an offset
110        * of 79.<br>
111        * Quadrant 3 and 4 indices are deduced from quadrant 1 and 2 by a symetry
112        * about x-axis and adding an offset. Quadrant N starts with index 1 + (N-1)*79.
113        *
114        * <p>EE- indices are deduced from EE+ by a symetry about (x,y)-plane (mirrored view).
115        * inconsistent with indices used in constructor EEDetId(int, int,int) in
116        * SCCRYSTALMODE</b>. Indices of constructor uses a symetry along y-axis: in principal
117        * considers the isc as a local index. The discrepancy is most probably due to a bug in
118        * implementation of this isc() method.
119        */
120       int isc() const ;
121
122       /** Gets crystal number inside SuperCrystal.
123        * Crystal numbering withing a supercrystal in each quadrant:
124        * \verbatim
125        *                       A y
126        *  (Q2)                 |                    (Q1)
127        *       25 20 15 10 5   |     5 10 15 20 25
128        *       24 19 14  9 4   |     4  9 14 19 24
129        *       23 18 13  8 3   |     3  8 13 18 23
130        *       22 17 12  7 2   |     2  7 12 17 22
131        *       21 16 11  6 1   |     1  6 11 16 21
```

```
132         *                       |
133         * ----------------------o---------------------------> x
134         *                       |
135         *       21 16 11  6 1   |     1  6 11 16 21
136         *       22 17 12  7 2   |     2  7 12 17 22
137         *       23 18 13  8 3   |     3  8 13 18 23
138         *       24 19 14  9 4   |     4  9 14 19 24
139         *       25 20 15 10 5   |     5 10 15 20 25
140         *  (Q3)                                      (Q4)
141         * \endverbatim
142         *
143         * @return crystal number from 1 to 25
144         */
145        int ic() const;
146
147        /** Gets the quadrant of the DetId.
148         * Quadrant number definition, x and y in std CMS coordinates, for EE+:
149         *
150         * \verbatim
151         *                 A y
152         *                 |
153         *          Q2     |    Q1
154         *                 |
155         *       ----------o---------> x
156         *                 |
157         *          Q3     |    Q4
158         *                 |
159         * \endverbatim
160         *
161         * @return quadrant number
162         */
163        int iquadrant() const ;
164
165        /** Checks if crystal is in EE+
166         * @return true for EE+, false for EE-
167         */
168        bool positiveZ() const { return id_&0x4000;}
169
170        int iPhiOuterRing() const ; // 1-360 else==0 if not on outer ring!
171
172        static EEDetId idOuterRing( int iPhi , int zEnd ) ;
173
174        /** Gets a compact index for arrays
175         * @return compact index from 0 to kSizeForDenseIndexing-1
176         */
177        int hashedIndex() const
178        {
179           const uint32_t jx ( ix() ) ;
180           const uint32_t jd ( 2*( iy() - 1 ) + ( jx - 1 )/50 ) ;
181           return (  ( zside()<0 ? 0 : kEEhalf ) + kdi[jd] + jx - kxf[jd] ) ;
182        }
183
184        /** Same as hashedIndex()
185         * @return compact index from 0 to kSizeForDenseIndexing-1
186         */
187        uint32_t denseIndex() const { return hashedIndex() ; }
188
189        /** returns a new EEDetId offset by nrStepsX and nrStepsY (can be negative),
190          * returns EEDetId(0) if invalid */
191        EEDetId offsetBy( int nrStepsX, int nrStepsY ) const;
192
193        /** returns a new EEDetId swapped (same iX, iY) to the other endcap,
194          * returns EEDetId(0) if invalid (shouldnt happen) */
195        EEDetId switchZSide() const;
196
197        /** following are static member functions of the above two functions
198          *  which take and return a DetId, returns DetId(0) if invalid
```

```
199         */
200       static DetId offsetBy( const DetId startId, int nrStepsX, int nrStepsY );
201       static DetId switchZSide( const DetId startId );
202
203       /** Checks validity of a dense/hashed index
204        * @param din dense/hashed index as returned by hashedIndex() or denseIndex()
205        * method
206        * @return true if index is valid, false otherwise
207        */
208       static bool validDenseIndex( uint32_t din ) { return validHashIndex( din ) ; }
209
210       /** Converts a hashed/dense index as defined in hashedIndex() and denseIndex()
211        * methods to a det id.
212        * @param din hashed/dense index
213        * @return det id
214        */
215       static EEDetId detIdFromDenseIndex( uint32_t din ) { return unhashIndex( din ) ; }
216
217       static bool isNextToBoundary(     EEDetId id ) ;
218
219       static bool isNextToDBoundary(    EEDetId id ) ;
220
221       static bool isNextToRingBoundary( EEDetId id ) ;
222
223       /** Gets a DetId from a compact index for arrays. Converse of hashedIndex() method.
224        * @param hi dense/hashed index
225        * @return det id
226        */
227       static EEDetId unhashIndex( int hi ) ;
228
229       /** Checks if a hashed/dense index is valid
230        * @see hashedIndex(), denseIndex()
231        * @param i hashed/dense index
232        * @return true if the index is valid, false otherwise
233        */
234       static bool validHashIndex( int i ) { return ( i < kSizeForDenseIndexing ) ; }
235
236       /** Checks validity of a crystal (x,y.z) index triplet.
237        * @param crystal_ix crystal x-index
238        * @param crystal_iy crystal y-index
239        * @param iz crystal z-index
240        * @see EEDetId(int, int, int, int) for index definition
241        * @return true if valid, false otherwise
242        */
243       static bool validDetId(int crystal_ix, int crystal_iy, int iz);
244
245       /** Returns the distance along x-axis in crystal units between two EEDetId
246        * @param a det id of first crystal
247        * @param b det id of second crystal
248        * @return distance
249        */
250       static int distanceX(const EEDetId& a,const EEDetId& b);
251
252       /** Returns the distance along y-axis in crystal units between two EEDetId
253        * @param a det id of first crystal
254        * @param b det id of second crystal
255        * @return distance
256        */
257       static int distanceY(const EEDetId& a,const EEDetId& b);
258
259
260       /** Gives supercrystal index from endcap *supercrystal* x and y indexes.
261        * @see isc() for the index definition
262        * @param iscCol supercrystal column number: supecrystal x-index for EE+
263        * @param iscRow: supecrystal y-index
264        * @return supercystal index
265        */
```

```
266       static int isc( int iscCol,   // output is 1-316
267                       int iscRow ) ; //
268
269       /** Lower bound of EE crystal x-index
270        */
271       static const int IX_MIN =1;
272
273       /** Lower bound of EE crystal y-index
274        */
275       static const int IY_MIN =1;
276
277       /** Upper bound of EE crystal y-index
278        */
279       static const int IX_MAX =100;
280
281       /** Upper bound of EE crystal y-index
282        */
283       static const int IY_MAX =100;
284
285       /** Lower bound of supercystal index as defined in isc()
286        */
287       static const int ISC_MIN=1;
288
289       /** Lower bound of crystal index within a supercrystal
290        */
291       static const int ICR_MIN=1;
292
293       /** Upper bound of supercystal index defined in isc()
294        * <p>Beware it differs from the number of supercrystals in one endcap,
295        * which is 312, because the numbering is not dense.
296        */
297       static const int ISC_MAX=316;
298
299       /** Upper bound of crystal index within a supercrystal
300        */
301       static const int ICR_MAX=25;
302
303       enum {
304          /** Number of crystals per Dee
305           */
306          kEEhalf = 7324 ,
307          /** Number of dense crystal indices, that is number of
308           * crystals per endcap.
309           */
310          kSizeForDenseIndexing = 2*kEEhalf
311       };
312
313       /*@{*/
314       /** function modes for EEDetId(int, int, int, int) constructor
315        */
316       static const int XYMODE        = 0;
317       static const int SCCRYSTALMODE = 1;
318       /*@}*/
319
320    private:
321
322       bool        isOuterRing() const ;
323
324       static bool isOuterRingXY( int ax, int ay ) ;
325
326       //Functions from B. Kennedy to retrieve ix and iy from SC and Crystal number
327
328       static const int nCols = 10;
329       static const int nCrys = 5; /* Number of crystals per row in SC */
330       static const int QuadColLimits[nCols+1];
331       static const int iYoffset[nCols+1];
332
```

```
333       static const unsigned short kxf[2*IY_MAX] ;
334       static const unsigned short kdi[2*IY_MAX] ;
335
336       int ix( int iSC, int iCrys ) const;
337       int iy( int iSC, int iCrys ) const;
338       int ixQuadrantOne() const;
339       int iyQuadrantOne() const;
340 };
341
342
343 std::ostream& operator<<(std::ostream& s,const EEDetId& id);
344
345 #endif
```

-- DavidCockerill - 23-Aug-2010

This topic: Main > EEDetId
Topic revision: r1 - 2010-08-23 - DavidCockerill