

Costruzione di Interfacce Lezione 25 Xml for dummies Parte 2

cignoni@iei.pi.cnr.it
<http://vcg.iei.pi.cnr.it/~cignoni>

La classe CIMd2

- ❖ Classe per caricare un oggetto quake2
 - ❖ Load(filename, texturename);
 - ❖ setAnim(start,end)
 - ❖ setFps()
 - ❖ Draw(doctime)
- ❖ l'oggetto di solito e' verticale sul piano xy con l'origine sotto i piedi.

Leggere e scrivere file in QT

- ❖ un file e' considerato come un particolare device I/O
- ❖ classe QIODevice (*astratta*)
 - ❖ un mezzo da cui o su cui si puo leggere/scrivere byte
 - ❖ La azioni importanti sono
 - ❖ open(int mode)/close()/flush()
 - ❖ at(Offset pos)
 - ❖ readBlock(char * data, Q_ULONG maxlen)/writeBlock()
 - ❖ readLine/getch/putch

QIODevice

- ❖ bool QIODevice::open (int mode)
- ❖ mode e' una combinazione di
 - ❖ IO_Raw *unbuffered file access*
 - ❖ IO_ReadOnly
 - ❖ IO_WriteOnly
 - ❖ IO_ReadWrite
 - ❖ IO_Append
 - ❖ IO_Truncate
 - ❖ IO_Translate *enables carriage returns and linefeed translation.*

Specializzazioni QIODevice

- ❖ Qbuffer
 - ❖ buffer in memoria di char
- ❖ QSocket e QSocketDevice
 - ❖ per l'accesso in rete.
- ❖ QFile: quello che ci interessa
- ❖ Come ci si scrive piu' ad alto livello?

QFile

- ❖ Cose utili:
 - ❖ QDir per gestire dir,
 - ❖ sempre con lo '/'
 - ❖ utile per manipolare pathnames
 - ❖ QDir::setCurrent(QString) settare la dir corrente cui fanno riferimento i path specificati successivamente (se non assoluti)

QTextStream

- ❖ classe per leggere/scrivere testo su un QIODevice
 - ❖ QTextStream::QTextStream (QIODevice * iod)
 - ❖ QTextStream::QTextStream (QString * str, int filemode)
- ❖ solito stile c++
 - ❖ << butta nello stream
 - ❖ >> legge dallo stream
 - ❖ flags e variabili per dire come si trasf i numeri
 - ❖ flags(int) con valori tipo *bin, oct, dec, fixedpos* ecc.
 - ❖ precision(int)

Esempio pratico

```
// il costruttore di QFile non apre il file!
QFile xfile("pippo.txt");

// cosi' come quello di QDir non cambia dir.
QDir xdir("tmp.txt");

// fa riferimento al path corrente dell'app...
xdir.mkdir("");

// ...che adesso viene cambiato
QDir::setCurrent(xdir.path());

// il file viene creato solo ora.
xfile.open(IO_WriteOnly);
QTextStream xstrm(&xfile);
xstrm << "prova";
xfile.close();
```

Dom model per XML

- ❖ Interfaccia dom per accedere e modificare file xml
- ❖ si costruisce una rappresentazione gerarchica (un albero!) del documento xml in memoria
- ❖ si lavora sulla rappresentazione in memoria
 - ❖ **leggere** ~ attraversare e interpretare l'albero in mem con una qualche visita
 - ❖ **scrivere** ~ creare l'albero xml in memoria visitando nel giusto ordine le nostre strutture dati e poi invocare un metodo save

QDomDocument

- ❖ la classe che rappresenta l'intero documento
- ❖ e' la radice dell'albero xml permette l'accesso a tutti gli elementi dell'albero
- ❖ e' una specializzazione di QDomNode

QDomNode

- ❖ Classe base
 - ❖ QDomDocument
 - ❖ QDomElement
 - ❖ QDomAttr
- ❖ metodi
 - ❖ accesso, inserzione/rimozione figli
 - ❖ che son sempre QDomNode
- ❖ si puo' sapere di che tipo e'
 - ❖ isDocument/ isElement/ isAttribute

QDomAttr

- ❖ Attributo di un elemento
 - ❖ name()
 - ❖ value()

```
<link href="http://www.trolltech.com" color="red" />

QDomElement e = ...
QDomAttr a = e.attributeNode( "href" );
// stampa "http://www.trolltech.com"
cout << a.value() << endl;
// change the node's attribute
a.setValue( "http://doc.trolltech.com" );
QDomAttr a2 = e.attributeNode( "href" );
// stampa "http://doc.trolltech.com"
cout << a2.value() << endl;
```

QDomElement

- ❖ rappresenta un elemento
 - ❖ tagName() che puo essere cambiato
 - ❖ zero o piu' attributes
 - ❖ QString attribute(name, defval)
 - ❖ QDomAttr attributeNode(name)

QDomDocument

- ❖ gli elementi non possono esistere scollegati da un QDomDocument
- ❖ QDomDocument contiene i metodi per **fabbricare** gli elementi
 - ❖ I nodi hanno un *owner* che il doc che li ha creati
 - ❖ tutte le classi QDom sono solo references
 - ❖ saranno cancellati quando sono cancellati tutti gli oggetti che le riferiscono e il document che li possiede
 - ❖ si possono importare nodi da altri documenti
 - ❖ importNode(..)

QDomDocument

- ❖ per settare il contenuto di un doc
 - ❖ setContent
 - ❖ si puo usare in input
 - ❖ QString
 - ❖ QByteArray
 - ❖ QIODevice
- ❖ Per salvare il contenuto di un doc
 - ❖ save(QTextStream) in uno stream
 - ❖ vale anche per singoli nodi
 - ❖ toString()

QDomDocument

```
QDomDocument doc( "mydocument" );
QFile file( "mydocument.xml" );
if ( !file.open( IO_ReadOnly ) ) return;
if ( !doc.setContent( &file ) ) {
    file.close();
    return;
}
file.close();
// print out the element names of all elements that are direct children
// of the outermost element.
QDomElement docElem = doc.documentElement(); // la radice

QDomNode n = docElem.firstChild();
while( !n.isNull() ) {
    QDomElement e = n.toElement(); // try to convert the node to an element.
    if ( !e.isNull() ) {
        cout << e.tagName() << endl; // the node really is an element.
    }
    n = n.nextSibling();
}
}
```

```
QDomDocument doc( "MyML" );
QDomElement root = doc.createElement( "MyML" );
doc.appendChild( root );

QDomElement tag = doc.createElement( "Greeting" );
root.appendChild( tag );

QDomText t = doc.createTextNode( "Hello World" );
tag.appendChild( t );

QString xml = doc.toString();
```

In pratica

- ❖ Per salvare uno scene graph in xml?
 - ❖ aggiungere un metodo XMLWrite ad ogni nodo

```
void CISGGroup::XMLWrite(FILE *fp)
{
    fprintf(fp, "<CSGGroup>\n");
    iterator i;
    for(i=Sons.begin(); i!=Sons.end(); ++i)
        (*i)->XMLWrite(fp);
    fprintf(fp, "</CSGGroup>\n");
}
```