Tutorial

Use CuteEntityManager::Entity

You need to link against the CuteEntityManager lib.

Inherit from entitiy

```
#include "../src/entity.h"

class Person: public CuteEntityManager::Entity { //you can also use using namespace CuteEntityMana
ger;
    Q_OBJECT  //really important!
    EM_MACRO(Person) //aswell important, too!
    Q_PROPERTY(QString firstName READ getFirstName WRITE setFirstName)
    Q_PROPERTY(QString familyName READ getFamilyName WRITE setFamilyName)

//... getter and setter
private:
    QString firstName;
    QString familyName;
}

class Group: public CuteEntityManager::Entity {
    Q_OBJECT
    EM_MACRO(Group)
    Q_PROPERTY(QList<QSharedPointer<Pupil>> pupils READ getPupils WRITE setPupils)
    Q_PROPERTY(QList<QSharedPointer<Person>> persons READ getPersons WRITE
               setPersons)
    Q_PROPERTY(QString name READ getName WRITE setName)
    Q_PROPERTY(QSharedPointer<Person> mainTeacher READ getMainTeacher WRITE
               setMainTeacher)
//...
}
```

Register Entity class

You must place the following code at a really early execution point:

```
#include "entityinstancefactory.h"
//...
CuteEntityManager::EntityInstanceFactory::registerClass<Person>();
CuteEntityManager::EntityInstanceFactory::registerClass<Group>();
```

If you don't do that, the entitymanager can't create dynamic instances of entities (especially needed for find method)

Create EntityManager instance

```
#include "../src/entitymanager.h"

//...

CuteEntityManager::EntityManager *em = new CuteEntityManager::EntityManager("QSQLITE", QDir::curre
ntPath() + "/db.sqlite", "", "", "", 0, true);
```

This will create a connection to a (new) SQLite file at currentPath with name "db.sqlite". You can also use ":memory" instead of a database on the filesystem.

Persist database tables

Option 1

```
QStringList inits = QStringList() << "Contact" << "Address" << "Person" <<
                        "Pupil" << "Group";
em->startup("0.1", inits,
               true);
```

This creates tables for the entity classes Contact, Address, Person, Pupil and Group. Once called, the method "startup()" creates a new DatabaseMigration in the backgroud(http://jenkins.danfai.de/jenkins/job/cuteentitymanager/doxygen/class_cute_entity_manager_1_1_database_migration.html). The attribute "version" of DatabaseMigration has the value "0.1".

Option 2

```
em->createTable("Person"); //if you don't want that relationTables will be created, please append
a parameter with false
```

Option 3

```
QSharedPointer<Entity> entity = QSharedPointer<Person>(new Person()).objectCast<Entity>();
em->createTable(entity);
```

Create, update and remove objects

See http://jenkins.danfai.de/jenkins/job/cuteentitymanager/doxygen/class_cute_entity_manager_1_1_entity_manager.html for more information.

Create

When any method have been called, the database will be changed.

```
QSharedPointer<Person> person = QSharedPointer<Person>(new Person());
person->setFirstName("Max");
person->setFamilyName("Mustermann");
em->create(person.objectCast<Entity>());
//or
em->save(person.objectCast<Entity>());
```

Update/Merge

```
//we reuse person from create
person->setFamilyName("Musterfrau");
em->merge(person.objectCast<Entity>());
//or
em->save(person.objectCast<Entity>());
```

Remove

```
em->remove(person.objectCast<Entity>());
em->remove<Person>(1); //id
```

Find objects

There are several opportunities:

```
QSharedPointer<Entity> entity = em->findById(1, "Person");
QSharedPointer<Person> entity = em->findById<Person>(1);
QList<QSharedPointer<Person> entities = em->findAll<Person>();
QHash<QString, QVariant> attributes = QHash<QString, QVariant>();
attributes.insert("firstName","Max");
QSharedPointer<Person> max = em->findEntitiesByAttributes<Person>(attributes); //gives the first P
erson with firstName "Max"
QList<QSharedPointer<Person>> multiMax = em->findAllEntitiesByAttributes<Person>(attributes); //gi
ves all Persons with firstName "Max"
QList<QSharedPointer<Person>> sqlMax = em->findEntitiesBySql<Person>("SELECT * FROM person WHERE f
irstName = \"Max\"); //same as findEntitiesByAttributes
```

Use the query api (to find objects)

See http://jenkins.danfai.de/jenkins/job/cuteentitymanager/doxygen/class_cute_entity_manager_1_1_query.html and http://jenkins.danfai.de/jenkins/job/cuteentitymanager/doxygen/class_cute_entity_manager_1_1_query_builder.html

Simple example:

```
    Query query = Query();
    query.appendWhere(e->getQueryBuilder()->like(QString("firstname"), QString("Ma"),
                  JokerPosition::BEHIND));
    query.appendWhere(e->getQueryBuilder()->andOperator());
    query.appendWhere(e->getQueryBuilder()->arbitraryOperator("<", "birthday",
                  QDate(1950, 10, 10)));
    query.setDistinct(true);
    query.appendOrderBy(OrderBy(QString("birthday"), Direction::SORT_DESC));
    query.setLimit(10);
    QList<QSharedPointer<Person>> list = e->find<Person>(query, true);
```

This would generate:

```
SELECT DISTINCT * FROM person WHERE firstname LIKE "Ma%" AND birthday < "1950-10-10" ORDER BY birt
hday DESC LIMIT 10;
```

Mapping relations

See http://jenkins.danfai.de/jenkins/job/cuteentitymanager/doxygen/class_cute_entity_manager_1_1_relation.html for more information about the "Relation" class.

Class Entitiy has this method:

```
virtual const QHash<QString, Relation> getRelations() const;
```

If you want to use relations, you have to override this method:

```
const QHash<QString, CuteEntityManager::Relation> Person::getRelations() const {
    QHash<QString,CuteEntityManager::Relation> hash = QHash<QString, CuteEntityManager::Relation>(
);
    //... relations
    return hash;
}
```

Many-to-One

```
//Relation to class Person from class Group
//Headerfile:
Q_PROPERTY(QSharedPointer<Person> mainTeacher READ getMainTeacher WRITE setMainTeacher)
QSharedPointer<Person> getMainTeacher() const;
void setMainTeacher(const QSharedPointer<Person> &value);
QSharedPointer<Person> mainTeacher;

//Implementation in getRelations:
hash.insert("mainTeacher", CuteEntityManager::Relation("mainTeacher", RelationType::MANY_TO_ONE));
```

One-To-Many

```
//Relation to class Group from class Person
//Headerfile:
Q_PROPERTY(QList<QSharedPointer<Group>> maintainedGroups READ getMaintainedGroups WRITE setMaintai
nedGroups)
QList<QSharedPointer<Group> > getMaintainedGroups() const;
void setMaintainedGroups(const QList<QSharedPointer<Group> > &value);
QList <QSharedPointer<Group>> maintainedGroups;
```

```
//Implementation in getRelations:
hash.insert("maintainedGroups", CuteEntityManager::Relation("maintainedGroups",RelationType::ONE_TO_MANY,QString("mainTeacher")));
//Third parameter is "mappedBy". This means that this relation is already defined in class Group (as many-to-one).
```

Many-To-Many

```
//Relation to class Address from class Person
//Headerfile:
Q_PROPERTY(QList<QSharedPointer<Address>> addresses READ getAddresses WRITE setAddresses)
QList<QSharedPointer<Address> > getAddresses() const;
void setAddresses(const QList<QSharedPointer<Address> > &value);
QList <QSharedPointer<Address>> addresses;

//Implementation in getRelations:
hash.insert("addresses", CuteEntityManager::Relation("addresses", RelationType::MANY_TO_MANY));
```

One-To-One

Special case of Many-To-One. You can use a syntax similiar to the one used in Many-To-One or/and One-To-Many.

Transient attributes

Class "Entitiy" has this method:

```
virtual const QStringList getTransientAttributes() const;
```

If you want transient attributes, you have to override this method in your entity class.

```
const QStringList Person::getTransientAttributes() const {
    QStringList list = QStringList();
    list.append("firstName"); // Attribute firstName won't persisted
    return list;
}
```

Inheritance

Inheritance Strategy

Entity has this method:

```
//header
virtual InheritanceStrategy getInheritanceStrategy() const;

//cpp
InheritanceStrategy Entity::getInheritanceStrategy() const {
    return InheritanceStrategy::JOINED_TABLE;
}
```

You can choose between PER_CLASS_TABLE and JOINED_TABLE. If Pupil inherits from Person and you choose JOINED_TABLE, the EntityManager will create 2 tables.
First table:
Person - e.g. with these columns: "id","firstName", "lastName"
Second table:
Pupil - e.g. with these columns: "id", "guardianNote" but NOT with "firstName" or "lastName".
Both tables will be joined (same primary keys) when you do INSERT/UPDATE/DELETE/SELECT.

If you choose PER_CLASS_TABLE, you can get these tables (if you let EntityManager create them for you):
First table:
Person - with columns "id", "firstName", "lastName"
Second table:
Pupil - with columns "id", "firstName", "lastName" and "guardianNote". Both tables are NOT joined. They don't have any relation.

Cascade Inheritance

Entity has this method:

```
//header
virtual bool isInheritanceCascaded() const;

//cpp
bool Entity::isInheritanceCascaded() const {
    return true;
}
```

For example: you have the class "Person" and the class "Pupil" (which inherits from Person). If this method returns true, a Person will also be deleted if you remove a Pupil object.

Use validators

There are several validators: CompareValidator, EmailValidator, NumberValidator, etc.

```
virtual QList<ValidationRule> validationRules() const;
```

You must override this method in your entity class.

```
QList<ValidationRule> Person::validationRules() const {
    QList<ValidationRule> rules = QList<ValidationRule>();
    rules.append(ValidationRule("length", {"firstName", "familyName"}, "min", 2));
    rules.append(ValidationRule("date", "birthday", "past", "", "min", QDate(1973, 1, 1)));
    return rules;
}
```

You can now call: em->validate(entity);
You can get the validation errors by entity->getErrors(); (returns QList<ErrorMsg>) or you can call entity->getErrorsAsString()
You can also check an entity for errors with entity->hasErrors()(returns bool)

Example:

```
    QSharedPointer<Entity> p1 = QSharedPointer<Person>(new Person("Thomas", "B", Person::Gender::M
ALE, "", QString(), QString(), QDate(), 0));
    //validation takes also place before save/create/merge
    qDebug() << "p1 valid:" << e->validate(p1);
```

Validate would give back a "false" cause the lastname is too short (only "B" was specified but string length must be longer than or equal to 2).
em->create(), em->merge() and em->save() will also call "validate" method. You can disable the validation in these method calls if you provide additional parameters. For additional information look into the Validators sample provided by the repository.

Additional information

```
    /**
     * You can define the following validators:
     * "compare" -> CompareValidator  (Params: ==, !=, <,>,<=,>=)
     * "default" -> DefaultValidator You must only provide a value. Param name can be empty.
     * "email" -> EmailValidator You can provide "full" param.
     *      It would check "Max Example<max.example@synlos.net>".
     *      Without this param it checks for "max.example@synlos.net"
     * "exists" -> ExistValidator Checks if a MANY_TO_ONE/ONE_TO_ONE Relation Entity already has a
 ID
     * "file" -> FileValidator (Params: mimeTypes, extensions, minSize, maxSize)
     * "image" -> ImageValidator !Only Available when the EM was compiled with QT += gui
     *      (Params: minWidth, maxWidth, minHeight, maxHeight)
     * "number" -> NumberValidator (Params: min, max)
     * "date" -> DateValidator (Params: future, past, min, max)
     * "required" -> RequiredValidator If specified it checks if a QVariant is null or a QString i
s empty
```

```
     * "unique" -> UniqueValidator - Not done, checks if a value was already used
     * "url" -> UrlValidator (Params: defaultSchemes) Checks for a valid URL(https://... or http:/
/...)
     * "pattern" -> PatternValidator You can define a param Name, but you only need to provide a P
aram value with the Pattern for the regular expression
     * "length"-> LengthValidator (Params: min, max)
     *
     * You must override virtual QList<ValidationRule> validationRules() const; of Entity Class
     * You can define a ValidationRule for example with:
     * ValidationRule("length", {"firstName", "familyName"}, "min", 2);
     * First is everytime the shortcut of the specific validator
     * Second must be a single attribute or a list of attributes
     * Then you can specify the params with values.
     */
```