Contents

[**Summary** 1](#_Toc24227779)

[**BaseFactoryRegistrations.h** 2](#_Toc24227780)

[**BaseFactory.h** 2](#_Toc24227781)

[**Shapes** 3](#_Toc24227782)

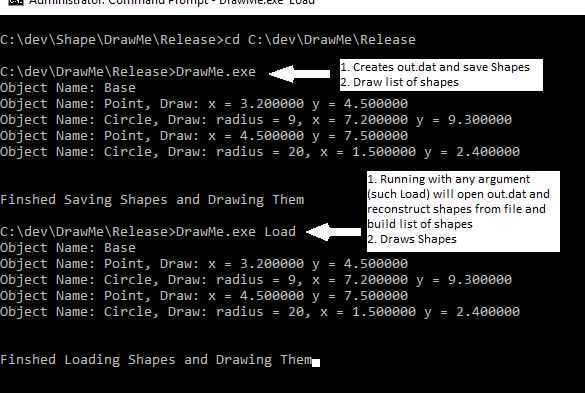
[**Archive** 4](#_Toc24227783)

# **Summary**

DrawMe.exe in the most generic way it could be done, without using boost Serialization nor Google Prototype mechanism (both and all solutions, register class type as it can’t be done otherwise in C++)

I’ve thought I’ve could overcome this by using type\_index, typeid and decltype in new versions of c++.

1. Shapes aren’t involved in the way they are saved or loaded and file format
2. Deserialization is the same for any Base Derived class using Class Factory, and Template solution



# **BaseFactoryRegistrations.h**

**Registers Base or Base Derived Class into a Generic Template Functor for Creating Right Object when requested according to Object Buffer Rescued from file**

**class ShapeFactory:public ShapeFactoryBase**

**class BaseFactoryRegistration**

**BaseFactoryRegistration(const char\* id) - Constructor**

1. General - Registers Base or Base Derived Class
2. Creates ShapeFactory Functor with Class T (Type) for converting buffer of chars and creating right Object
3. Registers in Class Factory Map id and ShapeFactoryBase Functor for enabling type less Class Factory

**Register - Mapping any type of Base Derived class to id name, and enable creating of ShapeFactory for creation the right Object**

#ifndef REGISTER\_CLASS

#define REGISTER\_CLASS(NAME, TYPE) \

**BaseFactoryRegistration<TYPE> objBase(NAME);**

#endif //

# **BaseFactory.h**

**class BaseFactory**

Type less Class Factory for creating the right shape object

Registers lambda function which creates ShapeFactoryBase functor and insert it to map according to id (Class name)

**class ShapeFactoryBase**

Calls the right ShapeFunctor using polymorphism, and creates the right Shape Object

# **Shapes**

//Register Shape for generic Class Factory (The Only Place where Shape Type is mentioned)

REGISTER\_CLASS("Base", Base);

REGISTER\_CLASS("Point", Point);

REGISTER\_CLASS("Circle", Circle);

I’m using polymorphism for Draw and due to vt I’m using virtual size for real object size

**class Base**

**class Point : public Base**

**class Circle : public Point**

# **Archive**

**class Archive**

void Add(shared\_ptr<Base> obj);

vector<shared\_ptr<Base>> vectorOfPoints;

**class InArchive: public Archive**

void Draw();

void Serialization(char \*fileName);

Writes Shape Size and Shape as Char Buffer into file(VM pointer increases in 4 bytes but doesn’t influence on my desrealization, it’s a matter of product to decide if to have virtual Draw or not)

**class OutArchive : public Archive**

void Draw();

void Deserialization(char \*fileName);

**Reconstructs shape from file in a generic way based on REGISTER\_CLASS shape**

**I’m using buffer twice, once for creating Base obj for retrieving name and second for creating the right type object according to name**

**//Read size buffer of chars**

**//Create Base Class from buffer**

Base \*pB = new Base(reinterpret\_cast<Base\*>(p));

**//Create generator Shape functor for creating right shape according to pB->name**

ShapeFactoryBasePtr obj = BaseFactory::get().getShape(pB->GetName().c\_str());

**/\*getShape retrieves a ShapeFactoryBasePtr obj functor according to pB->name.**

**obj uses virtual getShape with buffer of chars and uses shape functor generator to create the right shape\*/**

Add(shared\_ptr<Base>(obj->getShape(p)));

**//Delete buffer**

delete pB;