BaseLib2 Tutorial Series
Objects, Garbage Collection and Templates

André Neto

July, 2011

Outline

Garbage collection

- Good
  - Avoid misuse of pointers
  - Avoid memory leaks
  - Force good housekeeping
- At the price of
  - Having to use the BaseLib2 development model
  - Your object must be GarbageCollectable (see GarbageCollectable.h in Level1)
  - Create the objects using a special syntax (no **new** operator)
- The garbage collection mechanism tracks the number of references for a given object
  - When this goes to zero the object is automatically destroyed

GCReference

- The GCReference (**G**arbage**C**ollectableReference) is one of the key classes of BaseLib2
- It contains the pointer to your object
    - Tracks the number of instances
    - Deletes the object when no longer used
- Is capable of constructing an object by its class name (more on this later)
- Enables to query if its pointer object is valid
    - Although the GCReference exists it doesn't mean the pointer object is valid! (see example later)
- Provides the mechanism behind automatic object creation using data driven configurations (more on this in a later tutorial)
- The GCRTemplate (**G**arbage**C**ollectable**R**eferenceTemplate) is a templated version of GCReference

GCReference.h and GCRTemplate.h

Most important functions are

```
virtual bool ObjectLoadSetup( ConfigurationDataBase & info, StreamInterface * err, bool createOnly=False);
virtual bool ObjectSaveSetup( ConfigurationDataBase & info, StreamInterface * err);
virtual bool IsValid() const;
inline int32 NumberOfReferences() const;
inline GarbageCollectable* operator->() const;
inline bool operator==(const GCReference& reference) const;
inline bool operator!=(const GCReference& reference) const;
```

- Access these functions using the "." operator
- Access your object functions using the "->" operator
- Create new instance by using the flag GCFT_Create

GCReference Example (BaseLib2/Documentation/Tutorials/examples/GCReferenceExample1.cpp)

Example code

```
/**
* A simple class implementing GarbageCollectable so that it can be
* tracked by a GCReference
*/
class SimpleClass :  public GarbageCollectable{
public:
...
}
GCRTemplate<SimpleClass> simpleClassRef1;
//It should not be valid
if(!simpleClassRef1.IsValid()){
...
//Objects are created using GCFT_Create
GCRTemplate<SimpleClass> simpleClassRef2(GCFT_Create);
//This reference should be valid
if(simpleClassRef2.IsValid()){
...
//Now put another reference pointing to the same object
simpleClassRef1 = simpleClassRef2;
//Object variables and functions are access using the -> operator
simpleClassRef2->uniqueID = 123456789;
```

Creating objects by name

- Objects can be automatically created by the class name
- In order to use this functionality:
  - the class must inherit from Object (see Object.h)
  - 3 macros must be called (see ObjectMacros.h)
    - OBJECT_DLL, OBJECT_DLL_STUFF and OBJECTLOADREGISTER
  - the details behind these macros are quite complicated...
    - create a series of hidden functions based on the class name...
    - ...which add the class name to a database when the application starts...
    - ...can then be accessed in run-time to actually create and manage the class...
    - ...and get live information about it (class name, id, ...)
- This is the mechanism behind data-driven object creation which is another BaseLib2 very important feature

Object creation by name Example (BaseLib2/Documentation/Tutorials/examples/GCReferenceExample2.cpp)

Example code

```
//These macros create a series of hidden functions
//which allow to automatically create the object
OBJECT_DLL(SimpleClass)
class SimpleClass :  public GarbageCollectable, public Object{
OBJECT_DLL_STUFF(SimpleClass)
public:
...
};
//Usually a version id is set as the second argument
OBJECTLOADREGISTER(SimpleClass, "$Id:  GCReferenceExample2.cpp,v
1.1 2011/07/14 09:42:40 aneto Exp $")
//Try to create the object by name
GCRTemplate<SimpleClass> simpleClassRef1("SimpleClass");
//It should be valid
if(simpleClassRef1.IsValid()){
...
```

GCNamedObject and GCReferenceContainer

- The GCNamedObject (**G**arbage**C**ollectableObject) is also widely used
  - Stores the object name
- The GCReferenceContainer (**G**arbage**C**ollectableReferenceContainer) is a container for GarbageCollectable references
  - Find and retrieve objects by name and index or by using iterators

Most important functions of GCReferenceContainer are

```
inline GCReference Find(const char * name,GCFlagType recurse...);
inline GCReference Find( SearchFilterT<GCReference> * selector...);
inline bool Insert( GCReference gc, int position = -1);
inline GCReference Remove( const char * name, GCFlagType recurse...);
inline GCReference Remove( int index);
bool Iterate( IteratorT<GCReference> * iterator,...);
```

GCNamedObject and GCReferenceContainer Example (1) (BaseLib2/Documentation/Tutorials/examples/GCReferenceContainerExample.cpp)

Example code

```
class SimpleNamedObject :  public GCNamedObject{
OBJECT_DLL_STUFF(SimpleNamedObject)
public:
SimpleNamedObject(){
...
GCRTemplate<SimpleNamedObject> simpleNamedObj("SimpleNamedObject");
//Because it is a named object, we can associate a name to it
simpleNamedObj->SetObjectName("myObject");
//One way of verifying if an object is of given type
GCRTemplate<GCNamedObject> namedObj = simpleNamedObj;
//If this is valid (and in this case it is) then the reference is
valid...
if(namedObj.IsValid()){
...
//A SimpleClass if not of type GCNamedObject
GCRTemplate<SimpleClass> simpleClassObj2 = simpleNamedObj;
//So it should not be valid...
if(!simpleClassObj2.IsValid()){
...
```

GCNamedObject and GCReferenceContainer Example (2) (BaseLib2/Documentation/Tutorials/examples/GCReferenceContainerExample.cpp)

Example code

```
//Create a GCReferenceContainer to store our references
GCReferenceContainer gcRefContainer;
if(!gcRefContainer.Insert(namedObj)){
...
if(!gcRefContainer.Insert(simpleClassObj)) {
...
//Query the size of the container
CStaticAssertErrorCondition(Information, "The size of the gcRefContainer
is:  %d", gcRefContainer.Size());
//Look for objects based on name
GCRTemplate<SimpleNamedObject> myObject3 =
gcRefContainer.Find("myObject2");
//Look for objects based on index
//By using the IsValid the type of the object can be tested
int32 i=0;
for(i=0; i<gcRefContainer.Size(); i++){
    GCRTemplate<SimpleClass> obj = gcRefContainer.Find(i);
    if(obj.IsValid()){
...
```

AssertErrorCondition

- When using BaseLib2 objects you should log using the method AssertErrorCondition
  - More debugging information is automatically extracted (class name and object pointer)

AssertErrorCondition

Output using AssertErrorCondition:

```
|TM=4e3bf2ac|C=SimpleClass|O=0894b298|T=b785e6d0|E=00000001|D=Creating a
SimpleClass
```

Output using CStaticAssertErrorCondition:

```
|TM=4e3bf2ac|T=b785e6d0|E=00000001|D=Creating a SimpleClass
```

Training ideas (1/3)

1. Write a class named Shape
    1. This class must inherit from GCNamedObject
    2. It has a pure virtual method named Area which returns a float
    3. It has a pure virtual method named SetAttributes which receives an FString with the shape attributes separated by “,”
2. Write 3 classes, named Circle, Square and Rectangle which inherit from Shape
    1. Implement in each of these the Area and SetAttributes
        1. The Attributes will be separated by “,”
        2. The number of attributes will depend on the class (1 for circle and square, 2 for rectangle)

Training ideas (2/3)

1. Write an application which has a TCPServer listening on a given port
    1. Client will send the requests encoded as "order|ClassName|ObjectName|ShapeAttributes)"
    2. The order will be a single char which can be
        1. c, for create object
        2. d, for delete object
        3. l, for list objects
        4. e.g. c|Circle|myCircle|1
        5. e.g. c|Rectangle|myRectangle|2,3
        6. e.g. d|Rectangle|myRectangle
2. For each create order
    1. Create the object by ClassName
    2. Give the object the ObjectName
    3. Set the ShapeAttributes
    4. Add the created object to a GCReferenceContainer

Training ideas (3/3)

1. For each delete order
   1. Search for the object name in the GCReferenceContainer and remove it
2. For each list order
   1. Using an iterator, loop inside all the elements of the GCReferenceContainer and print
      1. Name
      2. Area