Programmers Manual for the Gizmo Software Development Kit (GizmoSDK)

Contents

1 GENERAL

1.1 What is GizmoSDK?

During the past years Saab Training Systems AB has developed applications for military combat simulation and training. Numerous times we reinvented the wheel in our application development and finally realized that we needed a general foundation architecture to solve these problems. We wanted out software developers to use a common set of API functions and that they should be able to communicate with design patterns and solutions common to all developers. We created a number of component libraries and APIs that are described below and put them together in a SDK that we call.

Gizmo Software Development Kit (*GizmoSDK*)

We always found out that we needed a platform abstraction API that isolated the different HW platform code solutions from our code so we could write once and run everywhere. We also found out that standard template library (STL) didn't work well in situations when we used shared libraries (.dlls), memory management was slow on Win32, exception management on PocketPC was not implemented, different programmers used different solutions on various platforms etc. so we tried to solve all these issues by a common layer API that forced many programmers in a group to write compatible code.

The solution was a library that we call **GizmoBase.** The library is a C++ layer that implements most needed features for memory management, error management, templates, reference pointers, file io etc.

In most of our systems we needed efficient real-time communication between applications distributed on a network. We needed a network solution that was faster then DCOM, CORBA or HLA and that was easy to use. That could work in-process just as good as between computers.

The solution we came up with was **GizmoDistribution**. By adding a distribution C++ code to your existing code, you can share objects and events between multiple computers or within a single process. It has the powerful control of objects and ownership that HLA uses but at the same time is mean and lean fast executing code.

The **GizmoDistribution** toolkit provides the distributed state of an application in real time.

We also needed a way to repeat the state like a tape recorder and playback the objects and events at a certain time with a certain speed. This functionality was solved by the **GizmoDynamics** Library. It uses a database with stored information about the **GizmoDistribution** internal states and can record and playback all activities and data that occurred on the **GizmoDistribution** network.

We also needed a fast and powerful API for 3D graphics so we could present large 2D and 3D maps, indoor and outdoor realistic environment with effects for snow, rain and daylight sun effects. There were lots of 3D APIs that could do this but none were flexible enough to be integrated in windowed applications with error management and progress information etc. We also wanted to use the latest graphics HW possibilities and the final solution was to write our own API. Our API, **Gizmo3D** is a high performance 3D Scene Graph and effect visualization C++ toolkit. It is similar to other scene graph engines such as Cosmo3D TM /OpenGL Performer TM/Inventor/VisKit/VTree TM but is a multi platform compatible API with very high performance. It also adds some state of the art functionality for effect presentations and graph interaction.

In advanced applications for 3D map presentation we need to present the map in accurate ways suited for specific customer in specific places. This means that we need to be able to manage local coordinate systems and metrics all over the world. The coordinate systems are described by local suitable projections and models of the world that fits the potato like world that are different in Australia than in Sweden etc. Different map formats are used in Norway and Austria etc. We also wanted to add state of the art measuring tools to our map presentation like sight calculations, diagrams, terrain utilities and feature selections etc. The solution for these problems was to add map features to some base classes in **Gizmo3D**.

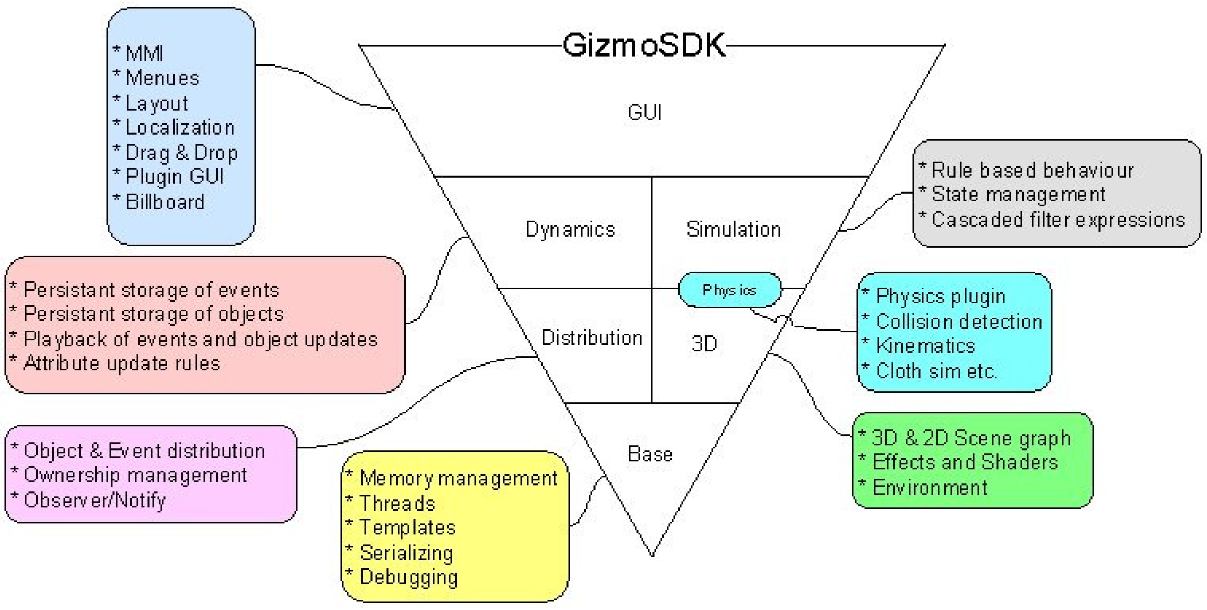
Now when we had the presentation and communication we needed to add simulation behaviors. A generic toolkit that could be used for evaluation of programmable behaviors of objects, a solution for fast and efficient generic simulation of object and we made a toolkit named **GizmoSimulation**. It provides the basic means for state machine management and filter pipelines.

All our application uses a common framework. We need a uniform look and feel for the GUI. We also want to use localized menus and texts, easy way to add plug-in GUI components that could be created at a later stage that the actual application was. All these nice to have features that makes a good and easy to use GUI is implemented in a common **GizmoGUI**. It is based by default on QT from TrollTech but any GUI implementation library can be used in combination with the GUI Framework utilities.

These are the libraries that form the **GizmoSDK**. They don't solve all tasks but they provide together a very complete base for all types of application development where many spent hours are saved in using them. Applications built up on **GizmoSDK** have a rock solid foundation to stand on!

1.2 Structure

The following picture describes the **GizmoSDK** structure and the relationship and dependency between the libraries.



At the bottom we find the **GizmoBase** library that all other libraries depend on. In general the top libraries depend on the lower libraries. Each library can be configured to include more or less features so they fit both lean platforms like Xbox and PocketPC as well as a high end PC Workstation.

As the Base library is the lowest level library, the **GizmoSDK** can run on all HW platforms that is supported by **GizmoBase**. The**GizmoBase** implementation provides a platform independency. Currently we support Linux, Irix, Win32 and Mac OS X/Iphone. Future versions might support BeOS, HP UX, Solaris, Xbox, PSx etc.

1.3 Rationale

We have gathered a number of important rationales that we experienced in our generic development cycle.

- GizmoSDK is a collection of software components that gives a uniform foundation for all our software development.

  We can develop software in a shorter time because many of our software design problems are already solved in GizmoSDK. We get fewer errors because the components have been tested in so many other applications already.

- The GizmoSDK components are uniform and cooperative in an open architechure.

  Developers working in a project can easy go on into another project or work with other tasks, as he will recognize the code structure and API in all situations.

- GizmoSDK simplifies large team development and supports process methodology standards.

- GizmoSDK is platform independent.

  Write once and run almost everywhere. The components are verified in our development process on all recommended platforms so they behave in the same way with threading, memory management, synchronization etc. that normally is very dependant on the selected platform.

- GizmoSDK is scaleable

2 GIZMOSDK OVERVIEW

2.1 Base Library

All GizmoSDK components are based on the Base library. Originally the base library was the base component of Gizmo3D witch is also a separate product from GizmoSDK. Therefore a lot of methods and nomenclature is taken from the Gizmo3D library and found in the Base or as it is also called the gzBase library.

It might be confusing to have e.g. gz prefix on all classes in Base but we need to keep the Gizmo3D and Base libraries uniform as a Gizmo3D product.

The purpose of the base toolkit is to provide a generic software foundation that can be used on all supported platforms. The toolkit provides type definitions, templates and utilities that do not need to be changed between different platforms.

2.2 Gizmo3D

The purpose of this API is to provide a high performance platform independent 3D engine. It also integrates with physics and other graphical simulators.

2.3 GizmoDistribution

The purpose of GizmoDistrubution is to distribute data between threads, processes, workstations and networks and enable applications and systems to share data in real-time. It implements a data-centric publish/subscribe design pattern that decouples publishers from subscribers. Events and objects states are distributed rather than providing a general purpose remote procedure call (RPC) based mechanism.

2.4 GizmoDynamics

The GizmoDynamics library provides a event recorder and a playback functionality for GizmoDistribution. You set the time and GizmoDynamics will update the internal state of GizmoDistribution objects to that recorded state.

2.5 GizmoSimulation

This library contains a state machine that is driven by events caused by triggers and generates actions as output.

2.6 GizmoGUI

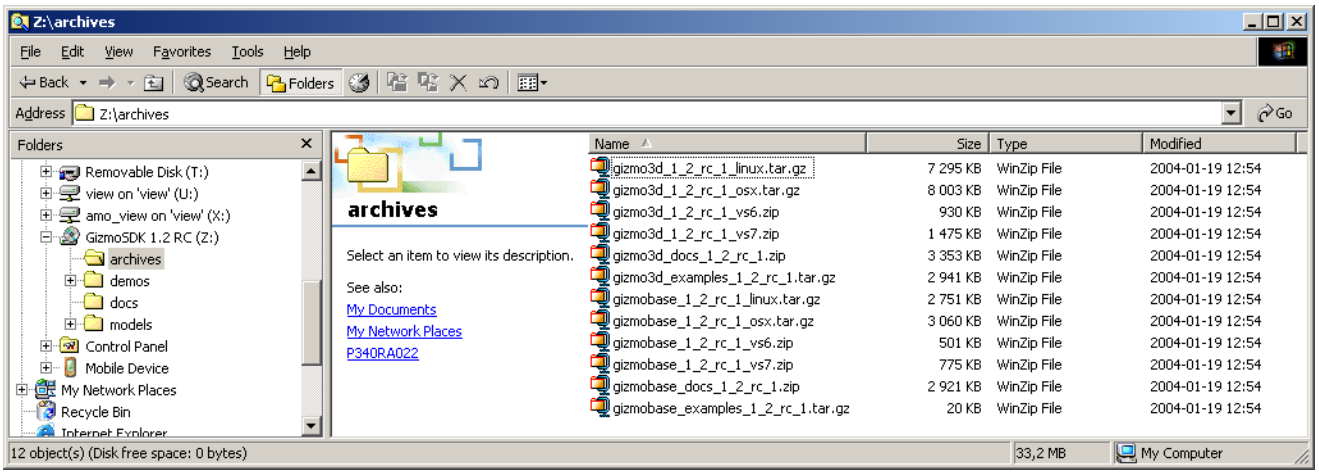
This library is a plugin FrameWork where you can write plugins with GUI content that fits in a generic application. The same application can be configured to have more or less componets activated.

2.7 GizmoTest

This is more of a framework for testing purposes. You can either run it through Microsoft test tool suite or you can use a stand alone application to do modular tests and unit tests in different plugins that you write.

3 INSTALLATION

First of all you should make sure you have all necessary files. The GizmoSDK distribution comes with a number of separate packages located on the SDK CD in the **archives** directory.


Z:\archives

Address Z:\archives

Folders: Removable Disk (T:), view on 'view' (U:), amo_view on 'view' (X:), GizmoSDK 1.2 RC (Z:), archives, demos, docs, models, Control Panel, Mobile Device, My Network Places, Recycle Bin, Internet Explorer

archives

Select an item to view its description.

See also:
My Documents
My Network Places
P340RA022

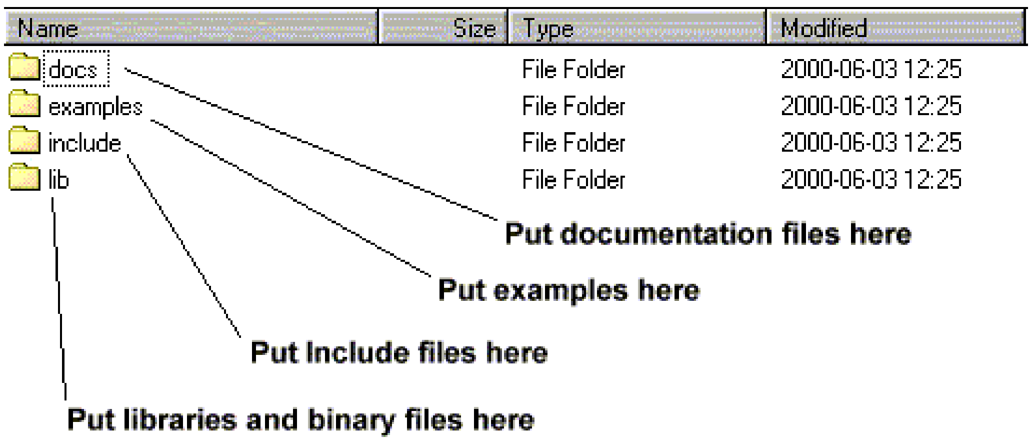
Name	Size	Type	Modified
gizmo3d_1_2_rc_1_linux.tar.gz	7 295 KB	WinZip File	2004-01-19 12:54
gizmo3d_1_2_rc_1_osx.tar.gz	8 003 KB	WinZip File	2004-01-19 12:54
gizmo3d_1_2_rc_1_vs6.zip	930 KB	WinZip File	2004-01-19 12:54
gizmo3d_1_2_rc_1_vs7.zip	1 475 KB	WinZip File	2004-01-19 12:54
gizmo3d_docs_1_2_rc_1.zip	3 353 KB	WinZip File	2004-01-19 12:54
gizmo3d_examples_1_2_rc_1.tar.gz	2 941 KB	WinZip File	2004-01-19 12:54
gizmobase_1_2_rc_1_linux.tar.gz	2 751 KB	WinZip File	2004-01-19 12:54
gizmobase_1_2_rc_1_osx.tar.gz	3 060 KB	WinZip File	2004-01-19 12:54
gizmobase_1_2_rc_1_vs6.zip	501 KB	WinZip File	2004-01-19 12:54
gizmobase_1_2_rc_1_vs7.zip	775 KB	WinZip File	2004-01-19 12:54
gizmobase_docs_1_2_rc_1.zip	2 921 KB	WinZip File	2004-01-19 12:54
gizmobase_examples_1_2_rc_1.tar.gz	20 KB	WinZip File	2004-01-19 12:54

12 object(s) (Disk free space: 0 bytes) 33,2 MB My Computer

Each component comes with three separate packages. XXX is the name for the specific package you should install.

The distribution package	**gizmoXXX_(*version*)_(*platform*).(*zip* or *gz*)**	Necessary to build component files, Select the right platform you want to use
The documentation package	**gizmoXXX_docs_(*version*).tar**	Platform independent documentation
The examples package	**gizmoXXX_examples_(*version*).tar**	Platform independent examples

The distribution package (**gizmoXXX_(*version*)_(*platform*).(*zip* or *gz*)**) contains the binary library files and include header files. The header files (*.h) shall be installed in the /include directory and the library files (*.so, *.dll , *.lib etc.) shall be installed in the /lib directory. The files shall be automatically put in the right directory when you unzip the distribution package to your selected **XXX** directory



Name	Size	Type	Modified
docs		File Folder	2000-06-03 12:25
examples		File Folder	2000-06-03 12:25
include		File Folder	2000-06-03 12:25
lib		File Folder	2000-06-03 12:25

Put documentation files here
Put examples here
Put Include files here
Put libraries and binary files here

The documentation package (**gizmoXXX_docs_(*version*).tar**) shall be unzipped into the docs directory. The online html help has the start page /docs/index.html. The GizmoBase online html help is included in Gizmo3D html help. If you use Gizmo3D you only need to download the Gizmo3D documentation to get all required html help.

The examples package (**gizmoXXX_examples_(*version*).tar**) shall be unzipped into the examples directory. Remember to unzip it with a preserved directory structure so you get the right directory structure and not mix all examples into one directory!

It is important to link against the correct libraries. All components use GizmoBase and then has to be linked against gzBase.

Gizmo3D must be linked against the following libraries:

gzBase	C++ library functions, platform abstractions
gzImage	Image formats like .bmp, .png etc
gzGraph	3D scene graph functions
gzDb	3D formats like .pfb , .flt etc.
gzSystem	Platform abstractions I/O like windows, joysticks etc.

GizmoDistribution must be linked against three libraries:

gzBase	
gzDistribution	local distribution
gzRemoteDistribution	remote distribution

The libraries can have extensions like _d for debug version, _dp for double precision.

If you are uncertain of how to use include files when using Gizmo3D you can include "gzGizmo3DLibrary.h" which will include all include files needed by the libraries. Each library above can use a subset of include files. In that case you only need to include "xxxLibrary.h" where xxx is the name of the library you need.

4 HOW TO BUILD EXAMPLES AND USE LIBS

The GizmoSDK API targets both Unix platforms as well as Win32 etc. The distribution contains examples where some are platform independent and some are not. There are a lot of important issues to know before you can use GizmoSDK successfully. They are not too complicated to understand.

4.1 General issues

The GizmoBase toolkit is free for all usage. If you want to use it in a test project etc. you can download a distribution from http://www.gizmosdk.com.

The other GizmoSDK components are free for non-commercial usage. If you want to use any of them in a test project etc. you can either download a test distribution that contains a 2-3 month evaluation temporary license key or email gizmosdk_info@sts.saab.se (for Gizmo3D the address is: gizmo3d_sales@sts.saab.se) with a request for a permanent license key. To obtain a permanent key you need to get your machine id with the example "**machid**" in the example directory. You have to send this key with the email together with a small note how you intend to use the toolkit. There will be an automatic registration form available on the web in the future.

Some other issues:

1. The binaries for the libraries use SINGLE_PRECISION floating point support. If you want DOUBLE_PRECISION you have to contact email: gizmo3d_support@sts.saab.se

2. The binaries for the libraries are default built as shared libs or dlls. If you want to use a static lib you need to contact Gizmo3D support or you need to have the Gizmo3D source code so you can compile it yourself

5 FAQ

6 LINKS

Here are some recommended links that you can take a look at

http://www.gizmosdk.com	The GizmoSDK Web Site
http://sts-websrv1/forum/	GizmoSDK internal development WEB

APPENDIX A - CODING GUIDELINES

Name space or prefix?

As we want to run the code on as many platforms as possible we have had to make some design decisions. E.g name space is not supported on older platforms, but we still wanted to use some kind of encoding to make the APIs to be separated from standard API calls. We selected to use prefix that shows what kind of API it is. E.g. the gz prefix is used to tell that the API call is taken from the **GizmoSDK** API family.

All global function calls and macros shall use ***gzXXX*** as identifier where gz is the API prefix. E.g if we have a global function called ***getNetworkIdentifier()*** and we will add it to the GizmoSDK API we shall name the function ***gzGetNetworkIdentifier()***. All global function calls shall start with small letters for the prefix. The prefix should be preferably two letters.

Macros

Macros that are dependant on compile settings shall have capital letters. E.g. there is a ***gzTrace*** that is always included as a macro. The ***GZTRACE*** macro is only present in the debug build.

Constants and defines

All constants and defines shall have the prefix YY_. E.g The Gizmo3D defines shall be named GZ_BLAH_IDENT etc.

Basic types

All APIs shall use the basic types defined in the Base API.
gzFloat, gzInt , gzString etc..

Class methods

All methods in a class shall start with a non capital letter and have capital letters for each name in the method. E.g. a method that gets the bets result shall be named as getTheBestResult().
When you have a get/set situation you should provide get/set methods for the class in a pair.

E.g. ***setTheLevel() / getTheLevel().***

When you have an enable/disable or read/write situation you should provide two methods with different in and return parameters

e.g. ***gzBool enableLevel()*** and ***gzVoid enableLevel(gzBool on)***

If you have default values of the set/get situation the parameters in the call shall have the default setting at creation time,

e.g. ***setTheLevel(gzULong level=5.0)*** if the level is 5.0 at creation time of the instance.

Class members

Use m_xXX on member vars to show that it is related to an instance member variable if it is non public. Skip the m_ if it is a public variable in an instance. It is not recommended to use public variable. Use get/set methods instead. However in some situation you have only data members and no methods (kind of struct). Then you can use public member variables.

Static member variables should use s_xXXXX as prefix.

DLL API management

Use export/import declarations on member methods in a class when you have instantiated templates in the class. If not you should try to use the import/export decl in the class definition to enhance readability,

e.g.

```
class gzClass
{
public:
     GZ_EXPORT gzBool doTheStuff();

private:

     gzList<gzMupp> m_muppisList;
};

// and when you have no templates

class GZ_EXPORT gzClass2
{
public:
     gzBool doTheStuff();
};
```

Brackets

Always use balanced bracket so you can tell what ending bracket belongs to what starting bracket above each other,

e.g.

```
if(a>b)
{
      doTheStuff();
}
else
{
      dontDoTheStuff();
}


switch(a)
{
      case 1 :   // Comment here

            doTheStuff();
            break;

      case 2 :   // Comment here what happens
      {
            doTheStuffInANewScope();
      }
}
```

APPENDIX B - CODING METHODOLOGY

There are certain guidelines for how to program a GizmoSDK code module. By recommending certain methods, we can make the software more uniform and well behaved in an integrated environment.

Here are a number of methods and rules (recommendations) that you can use to code your GizmoSDK code.

GZ_ASSERT usage

GZMESSAGE usage
Language macros

TBD !!

I just couldn't get this ready….