test Documentation

Release 1.0

Zhiyuan

Mar 10, 2017

Table of Content

CHAPTER 1

Table of Content

  - [[NPL Admin Site|AdminSiteFramework]]
  - [[Using TableDatabase|UsingTableDatabase]]
  - [[Using MySql|UsingMySql]]
  - [[HTTPS SSL Server|SecureDeployServer]]
- [[Resources]]
  - NPL Reference
  - ParaEngine Book
  - [[References]]
  - [[NPL Performance|NPLPerformance]]
  - [[ParacraftSnippets]]
- Join Us

CHAPTER 2

What is NPL?

NPL is short for neural parallel language.

- NPL is a general-purpose open-source language. Its syntax is 100% compatible with Lua.
- NPL provides essential functionality for building `3D/2D/Web/Server` applications.
- NPL provides rich C/C++ API and large collections of libraries written in NPL.
- NPL is a single language solution for advanced and interactive GUI, complex opengl/DirectX 3D graphics, scalable webserver and distributed software frameworks. It is cross-platform, high performance, extensible and debuggable.
- NPL is a language system initially designed to operate like the brain. Nodes and connections are ubiquitous; threading and networking logics are hidden from developers.
- NPL can mix user-mode `preemptive` and `non-preemptive` code. It gives you the concurrency of Erlang and speed of `Java/C++` in the same dynamic and weakly-typed language.

  See here for a list of projects written in NPL.

Facts You Should Know

- NPL runtime is written in C/C++. It can utilize the luajit compiler, which dynamically compiles script/byte code into native machine code, thus it can approach native C/C++ performance with some caveats.
- NPL runtime native API is a rich collection of C/C++ functions that is callable from NPL script. It includes networking, graphics, io, audio, assets management, and core NPL/ParaEngine infrastructure. NPL Reference contains its full documentation. However, 99% of time, we never use native API directly in NPL script.
- Instead, we use NPL libraries, which are pure NPL/lua script files, distributed open sourced or in a zip/pkg file. Over 1 million lines of fully-documented NPL script code are available for use. Try to search in the repository of NPL source code, before writing your own.
- NPL libraries are all written in `object oriented` fashion, code is organized in folders and table namespaces.

- Because NPL is written in C/C++, it is cross-platform, extensible and easy to integrate with other thirty-party tools. For examples, NPLRuntime is distributed with following built in plugins: bullet(a robust 3d physics engine), mono(C# scripting module with NPL API), mysql/sqlite(database engine), libcurl(a robust http/ssh client).
- There are two ways to use Lua, one is embedding, the other is extending it. NPL takes the former approach. For example, true preemptive programming would NOT be possible with the second approach. The embedding approach also allows NPL to expose all of its API and language features to other compiler, for example, each NPL runtime states can host Lua/Mono C#/C++ compiler states all together.

`Hello World` in NPL

Run in command line `npl helloworld.lua`. The content of `helloworld.lua` is like this:

```
print("hello world");
```

Now, there comes a more complicated helloworld. It turns an ordinary `helloworld.lua` file into a neuron file, by associating an `activate` function with it. The file is then callable from any NPL thread or remote computer by its NPL address(url).

```
NPL.activate("(gl)helloworld.lua", {data="hello world!"})
local function activate()
   local msg = msg;
   if(msg) then
      print(msg.data or "");
   end
end
NPL.this(activate);
```

If your code has `*.npl` file extension, you can use or extend NPL syntax via meta programming. For example, above code can be written with NPL syntax in `helloworld.npl` as below.

```
NPL.activate("(gl)helloworld.npl", {data="hello world!"})
this(msg){
   if(msg) then
      print(msg.data or "");
   end
}
```

Why Should I Use NPL?

You may write your next big project in NPL, if you meet any following condition, if not all:

- If you like truly dynamic language like lua.
- If you care about C/C++ compatibility and [[performance|NPLPerformance]].
- Coming from the C/C++ world, yet looking for a replacement language for java/C#, that can handle heavy load of 2d/3d graphical client or server applications, easier than C/C++.
- Looking for a scripting language for both client and server development that is small and easy to deploy on windows and other platforms.
- Looking for a dynamic language that can be just-in-time compiled and modified at run time.

- Mix user-mode `preemptive` and `non-preemptive` code for massive concurrent computation.

  Note: calling API between C++/scripting runtime, usually requires a context switch which is computationally expensive most language also involves type translation (such as string parameter), managed/unmanaged environment transition (garbage collection), etc. NPL/LUA has the smallest cost among all languages, and with luajit, type translation is even unnecessary. Thus making it ideal to choose NPL when some part of your responsive app needs to be developed in C/C++.

Background

NPL prototype was designed in 2004, which was then called 'parallel oriented language'. In 2005, it was implemented together with ParaEngine, a distributed 3d computer game engine.

Why a New Programming Language?

NPL is initially designed to write flexible algorithms that works in a multi-threaded, and distributed environment with many computers across the network. More specifically, I want to have a language that is suitable for writing neural network algorithms, 3d simulation and visualization. Lua and C/C++ affinity was chosen from the beginning.

Communicate Like The Brain

Although we are still unclear about our brain's exact computational model, however, following fact seems ubiquitous in our brain.

- The brain consists of neurons and connections.
- Data flows from one neuron to another: it is asynchronously, single-direction and without callback. Communication in NPL is the same as above. Each file can become a neuron file to receive messages from other neuron files. They communicate asynchronously without callback. As a result, no lock is required because there is no shared data; making it both simple and fast to write and test distributed algorithms and deploy software in heterogeneous environment.

Compare NPL With Other Languages

- `C/C++`:
  - Pros: They are actually the only popular cross-platform language. NPL Runtime is also written in C++.
  - Cons: But you really need a more flexible scripting language which are dynamically compiled and easier to use.
- `Mono C#`, `Java`:
  - Pros: Suitable for server side or heavy client. Rich libraries.
  - Cons: Deployment on windows platform is big. Writing C++ plugins is hard, calling into C++ function is usually slow. Language is strong typed, not really dynamic.
- `Node.js`, `Electron`:
  - Pros: Suitable for server side or heavy client. HTML5/javascript is popular.
  - Cons: Deployment on the client-side is very big. writing C++ plugins is very hard.
- `Python`, `Ruby`, `ActionScript` and many others:

  - Pros: Popular and modular
  - Cons: Performance is not comparable to C/C++; standard-alone client-side deployment is very big; syntax too strict for a scripting language. Some of them looks alien for C/C++ developers.

- `Lua`:
  - `Pros`: NPL is 100% compatible with it. Really dynamic language, highly extensible by C/C++, and syntax is clear for C/C++ developers.
  - `Cons`: Born as an embedded language, it does not have a standalone general-purpose runtime with native API support and rich libraries for complex client/server side programming. This is where NPL fits in.

- [[Concurrency Model|Concurrency]]: in-depth compare with erlang, GO, scala

References

Click the links, if you like to read the old history:

- NPL prototype doc in 2005
- A thesis on NPL/ParaEngine in 2005
- A book on ParaEngine in 2006

CHAPTER 3

Install NPL Runtime

You can redistribute NPLRuntime side-by-side with your scripts and resource files on windows/linux/iOS, etc. However, this section is only about installing NPL runtime to your development computer and setting up a recommended coding environment for NPL.

Install on Windows

Install from Windows Installer

It is very easy to install on windows. The win32 executable of NPL runtime is currently released in two forms, choose one that is most suitable to you:

- 32bits/64bits most recent development version (recommended for standalone application or module dev). Click to download
- Official stable version released in ParacraftSDK (recommended for paracraft mod dev). See below.

Paracraft is an IDE written completely in NPL. Follow the steps:

- Download ParacraftSDK
  - or you can clone it with following commands (you need git installed):

```
# clone repository & sub modules
git clone https://github.com/LiXizhi/ParaCraftSDK.git
cd ParaCraftSDK
git submodule init
git submodule update
# Go into the repository
cd ParaCraftSDK
# install paracraft
redist\paracraft.exe
# install NPLRuntime
NPLRuntime\install.bat
```

- `redist\paracraft.exe` will install the most recent version of `Paracraft` to `./redist` folder
  - if you failed to install, please manually download from Paracraft and copy all files to `redist` folder.
- Inside `./NPLRuntime/install.bat`
  - The NPL Runtime will be copied from `./redist` folder to `./NPLRuntime/win/bin`
  - The above path will be added to your `%path%` variable, so that you can run npl script from any folder.

Update NPL Runtime to latest version

- If you use NPLRuntime installer, simply download and reinstall it here
- If you use paracraftSDK, we regularly update `paracraft` (weekly) on windows platform, you can update to latest version of NPL Runtime following following steps.
  - run `redist\paracraft.exe` to update paracraft's NPL runtime in `redist` folder
  - run `NPLRuntime\install.bat` to copy NPL runtime executable from `redist` to `NPLRuntime/win/bin` folder
- Compiling from source code is always the latest version (recommended for serious developers), see next section.

On 64bits version

To build 64bits version of NPLRuntime, you need to build from source code (see next section), or download prebuild version from [[http://www.paracraft.cn]].

Our current build target for public users is still 32bits on windows platform.

> Please note that all compiled and non-compiled NPL/lua code can be loaded by both 32bits/64bits version of NPL runtime across all supported platforms (windows/linux/mac/android/ios) etc. However, if you use any C++ plugins (such as mysql, mono, etc), they must be built for each platform and CPU architecture.

Install on Windows From Source

> see .travis.yml or our build output for reference

- You need to download and build boost from source code. Version `1.55.0` and `1.60.0` are tested,but the latest version should be fine. Suppose you have unzipped the boost to `D:\boost`, you can build with following command.

```
bootstrap
b2 --build-type=complete
```

and then add a system environment variable called `BOOST_ROOT`, with value `D:\boost`(or where your boost is located), so that our build-tool will find boost for you.

- Download NPLRuntime source code, or clone it by running:

```
git clone https://github.com/LiXizhi/NPLRuntime.git
```

- Download cmake, and build either `server` or `client` version. They share the same code base but with different `cmakelist.txt`; `client` version has everything `server` has plus graphics API.
  - `Server` version: run `NPLRuntime/NPLRuntime/CMakeList.txt` in `NPLRuntime/bin/win32` folder (any folder is fine, but do not use the `cmakelist.txt` source folder), please refer to `./build_win32.bat` for details

   - `Client` version: run `NPLRuntime/Client/CMakeList.txt` in `NPLRuntime/bin/Client` folder (any folder is fine, but do not use the `cmakelist.txt` source folder)
     * **REQUIRED:** Install latest DirectX9 SDK in default location here.
     * There is also a cmake option called `ParaEngineClient_DLL` which will build NPLRuntime as shared library, instead of executable.
- If successful, you will have visual studio solution files generated. Open it with visual studio and build.
- The final executable is in `./ParaWorld/bin/` directory.

How to Debug NPLRuntime C++ source code

When you build NPLRuntime from source code, all binary file(dlls) are automatically copied to `./ParaWorld/bin/` directory. All debug version files have `_d` appended to the end of the dll or exe files, such as `sqlite_d.dll`, etc.

To debug your application, you need to start your application using the executable in `./ParaWorld/bin/`, such as `paraengineclient_d.exe`. You also need to specify a working directory where your application resides, such as `./redist` folder in `ParacraftSDK`. Alternatively, one can also copy all dependent `*_d.dll` files from `./ParaWorld/bin/` to your application's working directory.

Applications written in NPL are usually in pure NPL scripts. So you can download and install any NPL powered application (such as this one), and set your executable's working directory to it before debugging from source code. `ParacraftSDK` also contains a good collection of example NPL programs. Please see the video at [[TutorialWebSite]] for install guide.

Install on Linux

Click here to download precompiled NPL Runtime linux 64bits release.

Under linux, it is highly recommended to build NPL from source code with latest packages and build tools on your dev/deployment machine.

- Download NPLRuntime source code, or clone it by running:

```
git clone https://github.com/LiXizhi/NPLRuntime.git
git submodule init && git submodule update
```

- Install third-party packages and latest build tools.
  - see .travis.yml or our build output for reference (on debian/ubuntu). In most cases, you need `gcc 4.9 or above, cmake, bzip2, boost, libcurl`. Optionally, you can install `mysql, mono, postgresql` if want to build NPL plugins for them.
- Run ./build_linux.sh from the root directory. Be patient to wait for build to complete.

```
./build_linux.sh
```

Please note, you need to have at least 2GB memory to build with gcc, or you need to step up swap memory

- The final executable is in `./ParaWorld/bin64/` directory. A symbolic link to the executable is created in `/usr/local/bin/npl`, so that you can run NPL script from anywhere.

Install on MAC OSX

We use cmake to build under MAC OSX, it is very similar to `install on linux`, but you should use

```
./build_mac_server.sh
```

Currently, only server version is working on mac, client version is still under development.

Editors and Debuggers

You can use your preferred editors that support lua syntax highlighting. NPL Runtime has a built-in web server called [[NPL Code Wiki|NPLCodeWiki]], it allows you to edit and debug via HTTP from any web browser on any platform.

- [[NPL Code Wiki|NPLCodeWiki]]: build-in debugger and editor.

It is though highly recommended that you install NPL Runtime on windows, and use following editors:

- Visual Studio Community Edition: the free version of `visual studio`. This one is what I use.
- Visual Studio Code: a very good free/cross-platform editor based on `Atom`

We have developed several useful open-source NPL language plugins for visual studio products: In visual studio, select `menu::Tools::Extensions`, and search online for `npl` to install following plugins:

- NPL/Lua language service for visual studio: used by thousands of developers worldwide.
  - Download Source code
- NPL Debugger for visual studio: set break points and attach to running NPL process.
  - Download Source code
- NPL/ParaEngine Tools for visual studio: misc tools

Example Code

ParacraftSDK contains many example projects written in NPL.

- See ParacraftSDK wiki site for more details.
- You may now get your hand dirty by this tutorial or continue reading here.

CHAPTER 4

Projects Written in NPL

The following applications are completely written in NPL.

Paracraft

- Started: 2012-present
- Website: www.paracraft.cn
  - ParacraftSDK
  - Self Learning College

Paracraft is a standalone 3d animation software for everyone. It was my exploration of CAD(computer aided design) using the vision brain (i.e. your right brain). Paracraft is an extensible tool that everyone can learn to create videos and computer programs. I am using it to promote NPL language in self learning college to teach students programming.

Wikicraft

- Started: 2016-present
- Website: wikicraft.cn

Project is still in development. It is web site developed using NPL web framework.

Magic Haqi

- Started: 2009-2014 (still operating)
- Website: haqi.61.com
  - user forum

    - user videos

Magic Haqi is a free/paid 3D MMORPG published by taomee in November, 2009, allowing kids to play, create and share 3d worlds. It has over 5 million registered users and tens of thousands of user created works in China. The software is developed and owned by the developers of Paracraft, but has nothing to do with Paracraft. The initial version of paracraft was developed as a module in Magic Haqi. We have valid contract with its publisher taomee for their use of paracraft and its source code.

Magic Haqi2

- Started: 2012-2013 (still operating)
- Website: haqi2.61.com Same as Magic haqi.

Kids Movie Creator

- Started: 2006-2007
- Website: download link

Kids Movie Creator is a very old shareware released in 2006, allowing kids to create 3d world and make movies.

PaperStar

- Started: 2015-present
- Website: ps.61.com

This is a sandbox game developed by taomee.com using NPL and ParacraftSDK.

Projects Sponsored

Our investor will sponsor NPL related projects. If you have an idea, click here to get sponsored. The following is a list of projects being sponsored or open for recruitment.

- NPL 3D1
- NPL UI1
- :heart: NPL1
- :heart: CAD1
- NPL
- Paracraft EarthGIS
- :heart: GITNPL1
- Paracraft BMAX
- NPLMeta
- NPL CAD
- Paracraft

- NPL Replication
-
- 3D
- Npl Voxelizer finished
- NPL finished
- ParaEngine/NPL 3DMAC

Other projects

You are welcome to add your own software here

CHAPTER 5

NPL vs Lua

NPL embeds the small Lua/luajit compiler, and adds tons of native C/C++ API and new programming paradigms, making NPL a general purpose language/runtime.

Lua is an embedded language with the world fastest JIT compiler. The entire lua runtime is written in 6000 lines of code, making it very small and memory efficient to be embedded in another host language/runtime.

Host Languages vs Embedded Compilers

A high-level language contains its syntax, native API and libraries. All dynamic language runtimes must embed one or more compiler implementations in it. Compilers contains low-level implementation of compiling text code to bytecode and then to machine code.

For example, some version of JAVA, python, mono C#, objective C are first compiled into LLVM bytecode, and then the LLVM runtime compiles the bytecode to machine code. LLVM byte code is similar to lua or luajit byte code. Java/Python/MonoC# runtime embeds LLVM compiler in the same way as we embed lua/luajit in NPL. The difference is that LLVM is large and based on C/C++ syntax; but luajit is really small, and based on lua syntax. Languages like Java/Python/C# also have tons of host API in addition to LLVM in the same way of NPL hosting lua. This is why lua is designed to be an embedded language over which other host languages can be devised.

Facts about NPL vs Lua:

- NPL provides its own set of libraries written in NPL scripts, whose syntax is 100% compatible with Lua. This means that one can use any existing Lua code in NPL script, but the reverse is NOT possible.
- NPL runtime has over 700 000 lines of C/C++ NPL Host API code (not counting third-party library files), on which all NPL library files depends on. In NPL, we advise writing NPL modules in our recommend way, rather than using Lua's `require` function to load any external third-party modules.
- NPL library has over 1 000 000 lines of NPL script code covering many aspects of general-purpose programming. They are distributed as open source [[NPL packages|npl_packages]]. You are welcome to contribute your own NPL package utilizing our rich set of host API and libraries.

- NPL provides both non-preemptive and [[preemptive programming paradigm|Concurrency]] that is not available in Lua.
- NPL has built-in support for multi-threading and message passing system that is not available in lua.
- Finally, NPL loves Lua syntax. By default all NPL script files has `.lua` extension for ease of editing. Whenever possible, NPL will not deviate its syntax from lua. However, in case, NPL extends Lua syntax, NPL script file will use a different file extension. For example, `*.page`, `*.npl` are used just in case the script uses Non-Lua compatible syntax.

  NPL is not a wrapper of lua. It hosts lua's JIT compiler to provide its own programming API and language features, in addition to several other compilers, like mono C#, see below.

Compilers in NPL

NPL runtime state can run on top of three compiler frameworks simultaneously:

- Lua/Luajit compiler
- C/C++
- Mono/C# compiler Which means that you can write code with all NPL API and framework features in either of the above three compilers, and these code can communicate with one another with `NPL.activate` in the same NPL runtime process. Please see [[NPL Architecture|NPLArchitecture]] for details.

  Code targeting C/C++ should be pre-compiled. The other two can be compiled on the fly.

References

Figure 1.0: NPL Architecture Stack

CHAPTER 6

NPL Architecture

Figure 1.0: NPL Architecture Stack

Figure 2.0: NPL Scheduler

CHAPTER 7

Getting Started with NPL

The syntax of NPL is 100% compatible with lua. If you are unfamiliar with lua, the only book you need is

> Book: Programming in Lua

NPL/Lua Syntax in 15 minutes

The following NPL code is modified from [[http://learnxinyminutes.com/]]

```lua
-- Two dashes start a one-line comment.

--[[
     Adding two ['s and ]'s makes it a multi-line comment.
--]]

----------------------------------------------------
-- 1. Variables and flow control.
----------------------------------------------------

local num = 42;  -- All numbers are doubles.

s = 'string is immutable'
-- comparing two strings is as fast as comparing two pointers
t = "double-quotes are also fine"; -- `;` is optional at line end
u = [[ Double brackets
       start and end
       multi-line strings.]]
-- 'nil' is like null_ptr. It undefines t. When data has no reference,
-- it will be garbage collected some time later.
t = nil

-- Blocks are denoted with keywords like do/end:
while num < 50 do
  num = num + 1  -- No ++ or += type operators.
```

```
end

-- If clauses:
if (num <= 40) then
  print('less than 40')
elseif ("string" == 40 or s) then
  print('lua has dynamic type,thus any two objects can be compared.');
elseif s ~= 'NPL' then
  print('less than 40')
else
  -- Variables are global by default.
  thisIsGlobal = 5  -- variable case is sensitive.

  -- How to make a variable local:
  local line = "this is accessible until next `end` or end of containing script file"

  -- String concatenation uses the .. operator:
  print("First string " .. line)
end

-- Undefined variables return nil.
-- This is not an error:
foo = anUnknownVariable  -- Now foo == nil.

aBoolValue = false

-- Only nil and false are falsy; 0 and '' are true!
if (not aBoolValue) then print('false') end

-- 'or' and 'and' are short-circuited. They are like ||, && in C.
-- This is similar to the a?b:c operator in C:
ans = (aBoolValue and 'yes') or 'no'  --> 'no'

local nSum = 0;
for i = 1, 100 do  -- The range includes both ends. i is a local variable.
  nSum = nSum + i
end

-- Use "100, 1, -1" as the range to count down:
-- In general, the range is begin, end[, step].
nSum = 0
for i = 100, 1, -1 do
   nSum = nSum + i
end

-- Another loop construct:
repeat
  nSum = nSum - 1
until nSum == 0


----------------------------------------------------
-- 2. Functions.
----------------------------------------------------

function fib(n)
  if n < 2 then return 1 end
  return fib(n - 2) + fib(n - 1)
```

```
end

-- Closures and anonymous functions are ok:
function adder(x)
  -- The returned function is created when adder is
  -- called, and remembers the value of x:
  return function (y) return x + y end
end
a1 = adder(9)
a2 = adder(36)
print(a1(16))  --> 25
print(a2(64))  --> 100

-- Returns, func calls, and assignments all work
-- with lists that may be mismatched in length.
-- Unmatched receivers are nil;
-- unmatched senders are discarded.

x, y, z = 1, 2, 3, 4
-- Now x = 1, y = 2, z = 3, and 4 is thrown away.

function bar(a, b, c)
  print(string.format("%s %s %s", a, b or "b", c))
  return 4, 8, 15, 16, 23, 42
end

x, y = bar("NPL")  --> prints "NPL b nil"
-- Now x = 4, y = 8, values 15..42 are discarded.

-- Functions are first-class, may be local/global.
-- These are the same:
function f(x) return x * x end
f = function (x) return x * x end

-- And so are these:
local function g(x) return math.sin(x) end
local g; g  = function (x) return math.sin(x) end

-- Calls with one parameter don't need parenthesis:
print 'hello'  -- Works fine.


----------------------------------------------------
-- 3. Tables.
----------------------------------------------------

-- Tables = Lua's only compound data structure;
--          they are associative arrays.
-- Similar to php arrays or js objects, they are
-- hash-lookup dicts that can also be used as lists.

-- Using tables as dictionaries / maps:

-- Dict literals have string keys by default:
t = {key1 = 'value1', key2 = false}

-- String keys can use js-like dot notation:
print(t.key1)  -- Prints 'value1'.
```

```
t.newKey = {}  -- Adds a new key/value pair.
t.key2 = nil   -- Removes key2 from the table.

-- Literal notation for any (non-nil) value as key:
u = {['@!#'] = 'qbert', [{}] = 1729, [6.28] = 'tau'}
print(u[6.28])  -- prints "tau"

-- Key matching is basically by value for numbers
-- and strings, but by identity for tables.
a = u['@!#']  -- Now a = 'qbert'.
b = u[{}]     -- We might expect 1729, but it's nil:
-- b = nil since the lookup fails. It fails
-- because the key we used is not the same object
-- as the one used to store the original value. So
-- strings & numbers are more portable keys.

-- A one-table-param function call needs no parens:
function h(x) print(x.key1) end
h{key1 = 'Sonmi~451'}  -- Prints 'Sonmi~451'.

for key, val in pairs(u) do  -- Table iteration.
  print(key, val)
end

-- _G is a special table of all globals.
print(_G['_G'] == _G)  -- Prints 'true'.

-- Using tables as lists / arrays:

-- List literals implicitly set up int keys:
v = {'value1', 'value2', 1.21, 'gigawatts'}
for i = 1, #v do  -- #v is the size of v for lists.
  print(v[i])  -- Indices start at 1 !! SO CRAZY!
end
-- A 'list' is not a real type. v is just a table
-- with consecutive integer keys, treated as a list.

----------------------------------------------------
-- 3.1 Metatables and metamethods.
----------------------------------------------------

-- A table can have a metatable that gives the table
-- operator-overloadish behavior. Later we'll see
-- how metatables support js-prototypey behavior.

f1 = {a = 1, b = 2}  -- Represents the fraction a/b.
f2 = {a = 2, b = 3}

-- This would fail:
-- s = f1 + f2

metafraction = {}
function metafraction.__add(f1, f2)
  sum = {}
  sum.b = f1.b * f2.b
  sum.a = f1.a * f2.b + f2.a * f1.b
  return sum
end
```

```
setmetatable(f1, metafraction)
setmetatable(f2, metafraction)

s = f1 + f2  -- call __add(f1, f2) on f1's metatable

-- f1, f2 have no key for their metatable, unlike
-- prototypes in js, so you must retrieve it as in
-- getmetatable(f1). The metatable is a normal table
-- with keys that Lua knows about, like __add.

-- But the next line fails since s has no metatable:
-- t = s + s
-- Class-like patterns given below would fix this.

-- An __index on a metatable overloads dot lookups:
defaultFavs = {animal = 'gru', food = 'donuts'}
myFavs = {food = 'pizza'}
setmetatable(myFavs, {__index = defaultFavs})
eatenBy = myFavs.animal  -- works! thanks, metatable

-- Direct table lookups that fail will retry using
-- the metatable's __index value, and this recurses.

-- An __index value can also be a function(tbl, key)
-- for more customized lookups.

-- Values of __index,add, .. are called metamethods.
-- Full list. Here a is a table with the metamethod.

-- __add(a, b)                     for a + b
-- __sub(a, b)                     for a - b
-- __mul(a, b)                     for a * b
-- __div(a, b)                     for a / b
-- __mod(a, b)                     for a % b
-- __pow(a, b)                     for a ^ b
-- __unm(a)                        for -a
-- __concat(a, b)                  for a .. b
-- __len(a)                        for #a
-- __eq(a, b)                      for a == b
-- __lt(a, b)                      for a < b
-- __le(a, b)                      for a <= b
-- __index(a, b)  <fn or a table>  for a.b
-- __newindex(a, b, c)             for a.b = c
-- __call(a, ...)                  for a(...)

----------------------------------------------------
-- 3.2 Class-like tables and inheritance.
----------------------------------------------------

-- Classes aren't built in; there are different ways
-- to make them using tables and metatables.

-- Explanation for this example is below it.

Dog = {}                                   -- 1.

function Dog:new()                         -- 2.
```

```
  newObj = {sound = 'woof'}                -- 3.
  self.__index = self                      -- 4.
  return setmetatable(newObj, self)        -- 5.
end

function Dog:makeSound()                   -- 6.
  print('I say ' .. self.sound)
end

mrDog = Dog:new()                          -- 7.
mrDog:makeSound()  -- 'I say woof'         -- 8.

-- 1. Dog acts like a class; it's really a table.
-- 2. function tablename:fn(...) is the same as
--    function tablename.fn(self, ...)
--    The : just adds a first arg called self.
--    Read 7 & 8 below for how self gets its value.
-- 3. newObj will be an instance of class Dog.
-- 4. self = the class being instantiated. Often
--    self = Dog, but inheritance can change it.
--    newObj gets self's functions when we set both
--    newObj's metatable and self's __index to self.
-- 5. Reminder: setmetatable returns its first arg.
-- 6. The : works as in 2, but this time we expect
--    self to be an instance instead of a class.
-- 7. Same as Dog.new(Dog), so self = Dog in new().
-- 8. Same as mrDog.makeSound(mrDog); self = mrDog.

----------------------------------------------------

-- Inheritance example:

LoudDog = Dog:new()                           -- 1.

function LoudDog:makeSound()
  s = self.sound .. ' '                       -- 2.
  print(s .. s .. s)
end

seymour = LoudDog:new()                       -- 3.
seymour:makeSound()  -- 'woof woof woof'      -- 4.

-- 1. LoudDog gets Dog's methods and variables.
-- 2. self has a 'sound' key from new(), see 3.
-- 3. Same as LoudDog.new(LoudDog), and converted to
--    Dog.new(LoudDog) as LoudDog has no 'new' key,
--    but does have __index = Dog on its metatable.
--    Result: seymour's metatable is LoudDog, and
--    LoudDog.__index = LoudDog. So seymour.key will
--    = seymour.key, LoudDog.key, Dog.key, whichever
--    table is the first with the given key.
-- 4. The 'makeSound' key is found in LoudDog; this
--    is the same as LoudDog.makeSound(seymour).

-- If needed, a subclass's new() is like the base's:
function LoudDog:new()
  newObj = {}
  -- set up newObj
  self.__index = self
  return setmetatable(newObj, self)
end

-- Copyright: modified from http://learnxinyminutes.com/
-- Have fun with NPL/Lua!
```

How To Practice

We have developed NPL Code Wiki to help you learn and practice NPL. To launch NPL Code Wiki, you need to

- install Paracraft
- create an empty world or load online world like this one (HourOfCode)
- press F11 to launch [[NPL Code Wiki|NPLCodeWiki]].

You can then enter any NPL code in the console. See below:

Advanced Language Features

Lua/NPL is easy to learn, but hard to master. Pay attention to following concepts.

- Lua string is immutable, so comparing two strings is as fast as comparing two pointers.
- Lua has lexical scoping, use it to accelerate your code by exposing frequently used objects as local variables in its upper lexical scope.
- Always use local variables.
- NPL recommends object oriented programming, one can implement all C++ class features with lua tables, like class inheritance and virtual functions, etc.

CHAPTER 8

Hello World in NPL

This tutorial demonstrates

- how to setup a simple development environment
- write a program that output hello world
- explains the basic concept behind the scene

  This article is an aggregates of information you will find in this wiki guide. Skip this tutorial, if you have already read all related pages.

Setup Development Environment

For more information, see [[Install Guide|InstallGuide]].

- Download ParacraftSDK
  - or you can clone it by running:

```
git clone https://github.com/LiXizhi/ParaCraftSDK.git
```

- Make sure you have updated to the latest version, by running `./redist/paracraft.exe` at least once
- Run `./NPLRuntime/install.bat`
  - The NPL Runtime will be copied from `./redist` folder to `./NPLRuntime/win/bin`
  - The above path will be added to your %path% variable, so that you can run npl script from any folder.

To write NPL code, we recommend using either or both of following editors (and plugins):

- Visual Studio Community Edition: the free version of `visual studio`
  - In visual studio, select `menu::Tools::Extensions`, and search online for `npl` to install following plugins:
    - NPL/Lua language service for visual studio

    * NPL Debugger for visual studio

- Visual Studio Code: a cross-platform free editor based on Atom

Hello World Program

Run in command line `npl helloworld.lua`. The content of `helloworld.lua` is like this:

```
print("hello world");
```

You should see a file `./log.txt` in your current working directory, which contains the “hello world” text.

> If `npl` command is not found, make sure `./NPLRuntime/win/bin` is added to your search %path%; or you can simply use the full path to run your script (see below).

```
__YourSDKPath__/NPLRuntime/win/bin/npl helloworld.lua
```

Behind the Scenes

ParacraftSDK’s `redist` folder contains an installer of Paracraft. Running `Paracraft.exe` from redist directory will install the latest version of Paracraft, which ships with NPLRuntime. `./NPLRuntime/install.bat` copies only NPL runtime related files from redist folder to `./NPLRuntime/win/bin` and `./NPLRuntime/win/packages`

`./NPLRuntime/win/packages` contains pre-compiled source code of NPL libraries.

To deploy your application, you need to deploy both `bin` and `packages` folders together with your own script and resource files, such as

```
./bin/        :npl runtime 32bits
./bin64/      :npl runtime 64bits
./packages/   :npl precompiled source code
./script/     :your own script
./            :the working (root) directory of your application
```

In NPL script, you can reference any file using path relative to the working directory. So you need to make sure you start your app from the root directory.

Further Readings

ParacraftSDK contains many example projects written in NPL.

- See ParacraftSDK wiki site for more details.
- See [[Source Code Overview | SourceCodeOverview]]

CHAPTER 9

Video Tutorial: Create A Web Site

video

In this video tutorial, you will learn how to create a web site like this with NPL. The following topics will be covered:

- setup NPL web development environment
- NPL server page programming model
- Debugging and logs
- Use table database
- MVC frameworks for serious web development

  For complete explanation behind the scene, please read [[WebServer]]

Setup NPL web development environment

For more information, see [[Install Guide|InstallGuide]]. The following is only about setting up the recommended IDE under win32.

- Install Git so that you can call `git` command from your cmd line.
- Download ParacraftSDK
  - or you can clone it by running:

```
git clone https://github.com/LiXizhi/ParaCraftSDK.git
```

- Make sure you have updated to the latest version, by running `./redist/paracraft.exe` at least once
- Run `./NPLRuntime/install.bat`
  - The NPL Runtime will be automatically copied from `./redist` folder to `./NPLRuntime/win/bin`
  - The above path will be added to your `%path%` variable, so that you can run npl script from any folder.
- Download and install Visual Studio Community Edition: the free version of `visual studio`

  - In visual studio, select `menu::Tools::Extensions`, and search online for `npl` or `lua` to install following plugins: NPL/Lua language service for visual studio: this will give you all the syntax highlighting, intellisense and debugging capability for NPL in visual studio.

Create a New Visual Studio Project

NPL is a dynamic language, it does not require you to build anything in order to run, so you can create any type of visual studio project you like and begin adding files to it.

There are two ways to do it. Please use `the lazy way` but read `the manual one`.

The lazy way

- run `git clone https://github.com/tatfook/wikicraft.git`
- cd into the source directory and run `./update_packages.bat`
- open the `sln` file with visual studio
- run `start.bat`, you will see the npl server started.
  - visit [[http://localhost:8099]] for the main site.
  - visit [[http://127.0.0.1:8099]] for the NPL code wiki web site.
- rename the project to your preferred name and begin coding in the `./www` folder. (see next chapter)
- finally, see the `configure the project property` step in `the manual one` section.

The manual one

- create an empty visual studio project: such as a C# library or VC++ empty project.
- install the main npl package into your project root directory.

```
mkdir npl_packages
cd npl_packages
git clone https://github.com/NPLPackages/main.git
```

- configure the project property to tell `visual studio` to start NPLRuntime with proper command line parameters when we press `Ctrl+F5`.
  - For the external program: we can use either `npl.bat` or `ParaEngineClient.exe`, see below
  - external program: `D:\lxzsrc\ParaCraftSDKGit\NPLRuntime\win\bin\ParaEngineClient.exe`
  - command line parameters: `bootstrapper="script/apps/WebServer/WebServer.lua" port="8099" root="www/" servermode="true" dev="D:/lxzsrc/wikicraft/"`
  - working directory: `D:/lxzsrc/wikicraft/`
    - Note: the `dev` param and working directory should be the root of your web project directory.
- add main project at `npl_packages/main/*.csproj` to your newly created solution file, so that you have all the source code of NPL and NPL web server for debugging and documentation. **Reopen your visual studio solution file for NPL documentation intellisense to take effect.**
- create `www/` directory and add `index.page` file to it.
- Press `Ctrl+F5` in visual studio, and you will launch the web server.

  - visit [[http://localhost:8099/index.page]] in your browser.

NPL server page programming model

NPL web server itself is an open source application written completely in NPL. For its complete source code, please see `script/apps/WebServer/WebServer.lua` in the main npl_package.

So to start a website in a given folder such as `./www/`, we need to call

```
npl bootstrapper="script/apps/WebServer/WebServer.lua" port="8099" root="www/"
```

Normally, we also need to place a file called `webserver.config.xml` in the web root folder. If no config file is found, `default.webserver.config.xml` is used, which will redirect all request without extension to `index.page` and serve `*.npl`, `*.page`, and all static files in the root directory and it will also host NPL code wiki at [[http://127.0.0.1:8099]]. For more information, please see [[WebServer]]

Play with Mixed Code Web Programming

NPL server page is a mixed HTML/NPL file, usually with the extension `.page`. You can now play with it by creating new `*.page` file in `./www/` directory under visual studio. For more information, see [[NPLServerPage]].

Intellisense in visual studio

If you have installed NPL language service plugin and added main package csproj to your solution, you should already noticed intellisense in your visual studio IDE for all `*.lua`, `*.npl`, `*.page` files. Everything under `./Documentation` folder is used for auto-code completion. Please see `npl_packages/main/Documentation/`, you can also add your own intellisense by adding xml files to your own project's './Documentation' folder. For more information, please see [[NplVisualStudioIDE]].

Toggle Page Editing Mode

- One can toggle between two editing mode when writing `*.page` file in visual studio
  - right click the page file in solution manager, and select `open with...`
  - the default text editor will highlight NPL code in the page file, while all HTML text are greyed out
  - the HTML editor will highlight all html tags and all npl code are greyed out. (see below)

You can toggle between these mode depending on whether you are working on HTML or code.

How Does It Work?

At runtime, server page is preprocessed into pure NPL script and then executed.

For example

```
<html><body>
<?npl  for i=1,5 do ?>
   <p>hello</p>
<?npl  end ?>
</body></html>
```

Above server page will be pre-processed into following NPL page script, and cached for subsequent requests.

```
echo ("<html><body>");
for i=1,5 do
 echo("<p>hello</p>")
end
echo ("</body></html>");
```

When running above page script, `echo` command will generate the final HTML response text to be sent back to client.

Internals and Performance

Normally, a single NPL server can serve over 1000 dynamic page requests per seconds and with millions of concurrent connections (limited by your OS). It can serve over 40000 req/sec if it is very simple dynamic page without database query. See [[NPLPerformance]] and [[WebServer]] for details.

Unlike other preprocessed page language like `PHP`, database queries inside npl page file are actually invoked asynchronously, but written in the more human-friendly synchronous fashion. Please see `Use table database` chapter for details.

Debugging and logs

Most page errors are rendered to the HTTP stream, which is usually directly visible in your browser. For complete or more hidden errors, please keep `./log.txt` in your working directory open, so that you can read most recent error logs in it. Your error is usually either a `syntax` error or a `runtime` error.

To debug with NPL Code wiki

- Launch your website
- Right click on a line in visual studio and select `NPL set breakpoint here` in the context menu.
- The `http://127.0.0.1:8099/debugger` page will be opened.
  - Please note, your actual web site is at `http://localhost:8099`. They are two different websites hosted on the same port by the same NPL process, you can use the first `127.0.0.1` website to debug the second `localhost` website, because both web sites share the same NPL runtime stack/memory/code.

Please see [[DebugAndLog]] for general purpose NPL debugging.

Use table database

Table database is written in NPL and is the default and recommended database engine for your website. It can be used without any external dependency or configuration.

```
<html>
<body>
<?
-- connect to TableDatabase (a NoSQL db engine written in NPL)
db = TableDatabase:new():connect("database/npl/", function() end);

db.TestUser:findOne({name="user1"}, resume);
local err, user = yield(); -- async wait when job is done

if(not user) then
    -- insert 5 records to database asynchronously.
    local finishedCount = 0;
    for i=1, 5 do
        db.TestUser:insertOne({name=("user"..i)}, {name=("user"..i), password="1"},␣
↪function(err, data)
            finishedCount = finishedCount + 1;
            if(finishedCount == 5) then
                resume();
            end
        end);
    end
    yield(); -- async wait when job is done

    echo("<p>we just created "..finishedCount.." users in `./database/npl/TestUser.db`
↪</p>")
else
    echo("<p>users already exist in database, print them</p>")
end

-- fetch all users from database asynchronously.
db.TestUser:find({}, function(err, users)  resume(err, users); end);
err, users = yield(); -- async wait when job is done
?>

<?npl for i, user in ipairs(users) do ?>
    i = <?=i?>, name=<? echo(user.name) ?> <br/>
<?npl end ?>

</body>
</html>
```

Try above code. See [[UsingTableDatabase]] for details.

MVC frameworks for serious web development

There are many ways to write MVC framework with NPL. For robust web development, we will introduce the MVC coding style used in this sample web site.

The pros of using MVC framework is that:

- Making your website robust and secure
- Reuse the same code for both client/server side API calls.
- Writing less code for robust http `rest API`

It begins with a router page

All dynamic page requests are handled by a single file `index.page`, which calls `routes.page`.

The router page will check whether the request is a ajax rest API request or a normal page request according to its url. For example, every request url that begins with `/api/[name]/...` is regarded as an API request, and everything else is a normal page request.

If it is a ajax API request, we will simply route it by loading the corresponding page file in `models/[name].page`. In other words, if you placed a `user.page` file in `models` directory, you have already created a set of ajax HTTP rest API on the url `/api/user/...`.

Normally, all of your model file should inherit from a carefully-written `base` model class. So that you can define a robust data-binding interface to your low-level database system or anything else.

For example, the models/project.page, defines basic CRUD interface to `create/get/update/delete` records in the `project.db` database table. It also defines a custom ajax api at `project:api_getminiproject()` which is automatically mapped to the url `api/project/getminiproject/` in the router page (i.e. every method that begins with `api_` in the model class is a public rest API).

How you write your model class and define the mapping rules are completely up to you. The example website just gives you an example of writing model class.

As a website developer, you can choose to query model data on the client side with HTTP calls, or on the server side by directly calling model class method. Generally speaking, client side frameworks like `angular`, etc prefer everything to be called on the client side. We also favors client side query in most cases unless it is absolutely necessary to do it on the server side, such as content needs to be rendered on the server so that it is search-engine friendly.

Final Word

NPL used to be targeted at `3D/TCP server` development since 2005. NPL for Web development was introduced in early 2016, which is relatively new. So if there are errors or questions, please report on our issues.

Hope you enjoy your NPL programming~

CHAPTER 10

Video Tutorial: Create Paracraft Mod

video

In this video tutorial, you will learn how to create 3D applications with NPL. Make sure you have read our previous video [[TutorialWebSite]]. This is also going to be a very long video, be prepared and patient!

3D Graphics Powered By ParaEngine

NPL's graphics API is powered by a built-in 3D computer game engine called ParaEngine, which is written as early as 2005 (NPL was started in 2004). Its development goes side-by-side with NPL in the same repository. If you read the source code of NPLRuntime, you will notice two namespaces called `NPL` and `ParaEngine`.

In the past years, I have made several attempts to make the ParaEngine portion more modular, but even so, I did not separate it from the NPLRuntime code tree, because I want the minimum build of NPLRuntime to support all 3D API, just like JAVA, C# and Chrome/NodeJS does. However, if you build NPLRuntime with no `OpenGL/DirectX` graphics, such as on Linux server, these 3D APIs are still available to NPL script but will not draw anythings to your screen.

Introducing Our 3D World Editor: Paracraft

First of all, creating 3D applications isn't easy in general. Fortunately, NPL offers a free and open source NPL package called Paracraft, which is 3d world editor written in NPL.

Paracraft itself is also a standard alone software product for every one (including kids above 7 years old) to create 3D world, animated characters, movies and codes all in a single application. See [[http://www.paracraft.cn/]] for its official website. Actually there are hundreds of interactive 3D worlds, movies and games created by paracraft users without the need of any third-party software.

Fig. Screen shot of Paracraft

> Note: Yes, the look of it are inspired by a great sandbox game called `minecraft` (I love it very much), but using blocks is just the methodology we adopt for crafting objects almost exclusively. Blocks in

paracraft can be resized and turned into animated characters. The block methodology (rather than using triangles) allows everyone (including kids) to create models, animations, and even rigging characters very easily, and it is lots of fun. Remember you can still import polygon models from `MAYA/3dsmax`, etc, if you are that professional.

Let us look at some features of Paracraft.

- Everything is a block, entity or item, a block or entity can contain other items in it.
- Every operation is a command and most of them support undo and redo.
- There are items like scripts, movie blocks, bones, terrain brush, physics blocks, etc.
- Use NPL Code Wiki to view and debug.
- Network enabled, you can host your own private world and website in your world directory.

Creating Paracraft Plugin(Mod)

If you want to control more of the logic and looks of your 3D world, you need to write NPL code as professionally as we wrote paracraft itself. This is what this tutorial is really about.

Why Using Mod?

Because paracraft is also released as an open source NPL package, you can just modify the source code of Paracraft and turn it into anything you like or even write everything from scratch.

However, this is not the recommended way, because paracraft is maintained by us and regularly updated. If you directly modify paracraft without informing us, it may be very difficult to stay up to date with us. Unless you find a bug or wrote a new feature which we can merge into our main branch, you are recommended to use Paracraft Mod to create plugins to extend paracraft or even applications that looks entirely different from it.

Paracraft Mod or plugin system exposes many extension points to various aspects of the system. One can use it to add new blocks, commands, web page in NPL code wiki, or even replace the entire user interface and IO of paracraft.

_Fig. Above Image is an online game created via Paracraft Mod by one of our partners. _

You can see how much difference there is from paracraft.

Creating Your First Paracraft Mod

Install ParacraftSDK

- Install Git so that you can call `git` command from your cmd line.
- Download ParacraftSDK
  - or you can clone it by running:

```
git clone https://github.com/LiXizhi/ParaCraftSDK.git
```

- Make sure you have updated to the latest version, by running `./redist/paracraft.exe` at least once
- Download and install Visual Studio Community Edition: the free version of `visual studio`
  - In visual studio, select `menu::Tools::Extensions`, and search online for `npl` or `lua` to install following plugins: NPL/Lua language service for visual studio: this will give you all the syntax highlighting, intellisense and debugging capability for NPL in visual studio.

see [[Install Guide|InstallGuide]] and ParaCraftSDK Simple Plugin for more information.

Create `HelloWorld` Mod

- Run `./bin/CreateParacraftMod.bat` and enter the name of your plugin, such as `HelloWorld`. A folder at `./_mod/HelloWorld` will be generated.
- Run `./_mod/HelloWorld/Run.bat` to test your plugin with your copy of paracraft in the ./redist folder.

The empty plugin does not do anything yet, except printing a log when it is loaded.

Command line and runtime environment

If you open Run.bat with a text editor, you will see following lines.

```
@echo off
pushd "%~dp0../../redist/"
call "ParaEngineClient.exe" dev="%~dp0" mod="HelloWorld" isDevEnv="true"
popd
```

It specifies following things:

- the application is started via ParaEngineClient.exe in `./redist` folder
- the `dev` parameter specifies the development directory (which is the plugin directory ./_mod/HelloWorld/), basically what this means is that NPLRuntime will always look for files in this folder first and then in working directory (`./redist` folder).
- the `mod` and `isDevEnv` parameters simply tells paracraft to load the given plugin from the development directory, so that you do not have to load and enable it manually in paracraft GUI.

Install Source Code From NPL Package

All APIs of paracraft are available as source code from NPL package. Run `./_mod/HelloWorld/InstallPackages.bat` to get them or you can create `./_mod/HelloWorld/npl_packages` and install manually like this.

```
cd npl_packages
git clone https://github.com/NPLPackages/main.git
git clone https://github.com/NPLPackages/paracraft.git
```

You may edit `InstallPackages.bat` to install other packages, and run it regularly to stay up-to-date with git source.

> it is NOT advised to modify or add files in the ./npl_packages folder, instead create a similar directory structure in your mod directory if you want to add or modify package source code. Read [[npl_packages]] for how to contribute.

Setup Project Files

NPL is a dynamic language, it does not require you to build anything in order to run, so you can create any type of visual studio project you like and begin adding files to it. Open `ModProject.sln` with visual studio in your mod directory, everything should already have been setup for you. You can `Ctrl+F5` to run.

To manually create project solution file, follow following steps:

- create an empty visual studio project: such as a C# library and remove all cs files from it.
- add `npl_packages/main/*.csproj` and `npl_packages/paracraft/*.csproj` to your solutions, so that you have all the source code of NPL core lib and paracraft, where you can easily search for documentation and implementation.
- configure the project property to tell visual studio to start NPLRuntime with proper command line parameters when we press Ctrl+F5. The following does exactly the same thing as the `./Run.bat` in mod folder.
  - For the external program: we can use `ParaEngineClient.exe`, see below
  - external program: `D:\lxzsrc\ParaCraftSDKGit\redist\ParaEngineClient.exe`
  - command line parameters: `mod="HelloWorld" isDevEnv="true" dev="D:/lxzsrc/ParaCraftSDKGit/redist/_mod/HelloWorld/"`
  - working directory: `D:/lxzsrc/ParaCraftSDKGit/redist/_mod/HelloWorld/`
    - Note: the `dev` param and working directory should be the root of your mod project folder.

Please see my previous video tutorial for web server for more details about installing NPL language service for visual studio.

Understanding Paracraft Mod

When paracraft starts, it will load the file at `./Mod/_plugin_name_/main.lua` for each enabled plugin. It is the entry point of any plugin, everything else is up to the developer.

In our example, open the entry file `./_mod/HelloWorld/Mod/HelloWorld/main.lua` (see below). Because development directory is set to `_mod/HelloWorld` by startup command line, the relative path of above file is `./Mod/HelloWorld/main.lua`. This is why Paracraft can find the entry file.

```
local HelloWorld = commonlib.inherit(commonlib.gettable("Mod.ModBase"),commonlib.
↪gettable("Mod.HelloWorld"));

function HelloWorld:ctor()
end

-- virtual function get mod name
function HelloWorld:GetName()
    return "HelloWorld"
end

-- virtual function get mod description
function HelloWorld:GetDesc()
    return "HelloWorld is a plugin in paracraft"
end

function HelloWorld:init()
    LOG.std(nil, "info", "HelloWorld", "plugin initialized");
end

function HelloWorld:OnLogin()
end

-- called when a new world is loaded.
function HelloWorld:OnWorldLoad()
end

-- called when a world is unloaded.
```

```
function HelloWorld:OnLeaveWorld()
end

function HelloWorld:OnDestroy()
end
```

Most plugin will register a class in the "Mod" table, like above. The class usually inherits from `Mod.ModBase`. The class will be instantiated when paracraft starts, and its virtual functions are called at appropriate time.

For example, if your plugin contains custom commands or block items, you can register them in the `init()` method.

How to write plugin

There are three recommended ways to extend paracraft

- create your own command: command is a powerful way to interact with the user. The rule of thumb is that **before you have any GUI dialog, consider writing a command first**. Command is easy to test, integrate, and capable of interacting with other commands. Graphic User Interface(GUI) is good, however, it can only interact with humans in strict ways, commands can interact with both human and other commands or programs.
- create custom items: custom items include entities, items, or blocks. This ensures that your code is modular and independent. The best way to add new functions to paracraft is by adding new block or item types. Consider the color block and movie block in paracraft, which contains both GUI and complex logics.
- add filters: filters is a useful way to inject your code at predefined integration points in Paracraft.

The least recommended way to extend paracraft is to clone the source code files of paracraft to development directory and modify them. Because plugin directory is searched before the main paracraft source code pkg file, it is possible to overwrite source code of paracraft in this way, but you are highly decouraged to do so. If you are requesting a feature that can not be added via the recommended ways, please inform the developers of paracraft to add a new filter for you, instead of modifying our source code directly in your plugin.

Extending Command

see `_mod/Test/Mod/Test/DemoCommand.lua`

Extending GUI

see `_mod/Test/Mod/Test/DemoGUI.lua`

Extending Item

TODO:

Extending NPL Code Wiki

Users of Paracraft knows about `NPL Code Wiki`, if you want to use latest web technology (HTML5/js) to create user interface that can interact with your 3D application, you can extend the NPL code wiki.

Now let us create a new web page and add a menu item in NPL Code wiki

First of all, you need to place your server page file in NPL code wiki's web root directory. So we can create a file at `./_mod/HelloWorld/script/apps/WebServer/admin/wp-content/pages/HelloWorld.page`

its content may be

```
<h1>Hello World from plugin!</h1>
```

In the `./_mod/HelloWorld/Mod/HelloWorld/main.lua` file's init function, we add a menu filter to attach a menu item.

```
function HelloWorld:init()

    -- add a menu item to NPL code wiki's `Tools:helloworld`
    NPL.load("(gl)script/apps/WebServer/WebServer.lua");
    WebServer:GetFilters():add_filter( 'wp_nav_menu_objects', function(sorted_menu_
↪items)
        sorted_menu_items[sorted_menu_items:size()+1] = {
            url="helloworld",
            menu_item_parent="Tools",
            title="hello world",
            id="helloworld",
        };
        return sorted_menu_items;
    end);
end
```

Now restart paracraft, and press F11 to activate the NPL code wiki, you will notice the new menu item in its tools menu.

Documentation

The best documentation is the source code of paracraft which is in the paracraft package and several plugin demos in the `./_mod/test` folder. Since code is fully documented, use search by text in source code for documentations, as if you are googling on the web.

Debugging

For detailed information, please see NPL Debugging and Logs

- Use `./redist/log.txt` to print and check log file.
- press `F12` to bring up the buildin debugger.
- use `Ctrl+F3` to use the MCML browser for building GUI pages.
- use GUI debuggers for NPL in visual studio.

Plugin Deployment

To deploy and test your plugin in production environment, you need to package all the files in your plugin's development directory (i.e. `_mod/HelloWorld/*.*`) into a zip file called `HelloWorld.zip`, and place the zip file to './redist/Mod/HelloWorld.zip'. It is important that the file path in the zip file resembles the relative path to the development directory (i.e. `Mod/HelloWorld/main.lua` in the `HelloWorld.zip`).

Once you have deployed your plugin, you can start paracraft normally, such as running `./redist/Paracraft.exe`, enable your plugin in Paracraft and test it.

You can redistribute your plugins in any way you like. We highly recommend that you code and release on github and inform us (via email or pull request) if you have just released some cool plugin. We are very likely to fork your code and review it.

Further Readings

- Check out all sample plugins in `./_mod` folder.
- Check out all the SDK video tutorials by the author of Paracraft on the website.
- Read NPL Runtime's official wiki

Why are you doing this?

Lots of people and even our partners asked me why are you doing this `NPL/Paracraft/ParaEngine` thing and make all of them open source? To make the long story short, I want to solve two problems:

- Create a self-learning based educational platform so that everyone can learn computer science as my own childhood does.
  - We hope in 7 years, this could be the best present that we can give to our children. Right now, I really cannot find any material even close to it on the Internet.
- Create a online 3D simulated virtual world in which we can test new artificial intelligence algorithms with lots of humans. Using blocks is obviously the most economical way.

Thanks to our generous investor at `Tatfook`, we are able to sponsor more independent programmers and support more teams and partners using NPL to fulfill the above two goals. Please read this if you are interested in joining us and be sponsored.

CHAPTER 11

NPL Source Code Overview

This is a brief overview of NPL source code. The best way to learn NPL is to read its source code.

All NPL related source code is distributed as packages.

Two important packages for NPL and paracraft developement is at:

- main package: NPL standard library source code
- paracraft package: paracraft source code

```
cd npl_packages
git clone https://github.com/NPLPackages/main.git
git clone https://github.com/NPLPackages/paracraft.git
```

When you program, you can search by text in all NPL source code.

Source code overview

Core Library Files

- script/ide/
- script/ide/commonlib.lua: common include file for frequently used libraries without GUI.
- script/ide/System: Core library
- script/ide/STL: Like C++ stl, simple dData structures
- script/ide/timer.lua: time class
- script/ide/math: vector, matrix, etc.

Object Oriented

- script/ide/oo.lua: class inheritance implementation
- script/ide/System/Core/ToolBase.lua: a powerful base class with signal/property support.

Web Server Source Code

- script/apps/WebServer: implementation of a web server (like Apache) in NPL.
- script/apps/WebServer/WebServer.lua: entry file
- script/apps/WebServer/admin: A php-like web site framework in NPL

Paracraft Source Code

Lots of code here. If you are developing Paracraft plugins or 3d client applications in NPL. It is important to read source here.

- script/apps/Aries/Creator/Game: main source code of Paracraft
- [script/apps/Aries/Creator/Game/Areas](https://github.com/NPLPackages/paracraft/tree/master/script/apps/Aries/Creator/Game/Areas): desktop GUI
- script/apps/Aries/Creator/Game/Entity: All movable entity types
- script/apps/Aries/Creator/Game/blocks: All block types
- [script/apps/Aries/Creator/Game/Commands](https://github.com/NPLPackages/paracraft/tree/master/script/apps/Aries/Creator/Game/Commands): In-Game Commands
- [script/apps/Aries/Creator/Game/Tasks](https://github.com/NPLPackages/paracraft/tree/master/script/apps/Aries/Creator/Game/Tasks): Commands with GUI
- [script/apps/Aries/Creator/Game/GUI](https://github.com/NPLPackages/paracraft/tree/master/script/apps/Aries/Creator/Game/GUI): editor GUIs
- [script/apps/Aries/Creator/Game/Network](https://github.com/NPLPackages/paracraft/tree/master/script/apps/Aries/Creator/Game/Network): Paracraft server implementation
- [script/apps/Aries/Creator/Game/Shaders](https://github.com/NPLPackages/paracraft/tree/master/script/apps/Aries/Creator/Game/Shaders): GPU shaders
- [script/apps/Aries/Creator/Game/Mod](https://github.com/NPLPackages/paracraft/tree/master/script/apps/Aries/Creator/Game/Mod): Plugin interface
- script/apps/Aries/Creator/Game/SceneContext: User input like keyboard and mouse

CHAPTER 12

NPL Code Wiki

NPL Code Wiki is a web site which can be served directly from your local computer using NPL Runtime. It provides you a way to communicate with NPL Runtime via any web browser.

- NPL Code Wiki is designed to help people learn and practice NPL programming.
- It provides a way to inspect or debug NPL Runtime via HTTP protocol.
- Providing an interactive console to edit/debug NPL code at runtime.

Start Code Wiki

Start With NPL Runtime

Run from console

```
npl script/apps/WebServer/WebServer.lua
```

The default web address is [[http://127.0.0.1:8099/]]. For more information, please see [[WebServer]] section.

Start With Paracraft

To launch NPL Code Wiki, you need to

- install Paracraft
- create an empty world
- press F11 to launch. NPL Code wiki will be opened using your default web browser.

You can then enter any NPL code in the console. See below:

NPL Code Wiki Source Code is at script/apps/WebServer

Alternatively, you can start it via following code. Press `F12` in any world in Paracraft to enter code.

```
NPL.load("(gl)script/apps/WebServer/WebServer.lua");
WebServer:Start("script/apps/WebServer/admin");
```

NPL Debugger

NPL debugger is a HTTP protocol based debugger. It can be opened from `menu::tools::NPL debugger`, or with `Ctrl+Alt+I` in paracraft.

Because it is based on HTTP and served from the same NPL runtime you are debugging, you can debug your running application from any modern web browser. For example, you can debug apps running on linux server or mobile device remotely from a windows client. However, you must make sure that the apps are started with full source code available in its running directory.

How to Attach

There is no need to start your app in debug mode. Instead, just launch NPL code wiki, and follow the steps:

- In the web page, open your script files and press F9 or click the left margin of the line to set breakpoints.
- Click `Attach` button to begin debugging.
  - You can then press F10, F11, Shift+F11 or press the buttons to step through your code.
  - In the watcher window, you can evaluate any code or variable. Just type the variable name or code and click `evaluate`
- When debugger is attached, your program may run pretty slow, so once you are finished, click `Stop` button to detach.

Limitations

Currently only the main thread can be debugged, if you want to debug other thread, you can write your own debugger in NPL, the complete source code of the NPL debugger web site is only 400 lines.

NPL Code Editor

This is a multi-tab code editor for NPL. It can be opened from menu::view::code editor.

- Enter file name and press enter to open the file.
- Click the `sync` button on left top corner to navigate to the active document's directory.
- Even if you close your computer, your last opened files will be remembered when you open it again.
- Right click on the file name tab for a list of commands, such as `open containing folder`, `reload`, etc.
- Modified files will be marked with `*` in its tab, and need to be confirmed before closing it.
- `Ctrl+S` to save to disk.
- Filter files as you type in the left top's directory text box.

Object Inspector

Click `Menu::View::ObjectInspect` to open `object inspector`. Here you can view and edit all objects in NPL Runtime.

Once you edited a property, a command string is generated on top of the property window. The command string is equivalent to your last operation. In above image, the command string to change the window title is:

```
/property -all WindowText Paracraft
```

> The server-side NPL code that realized the `Object Inspector` page can be viewed by clicking the `view source` link on the bottom of the web page. This provides you a way to study the NPL implementation of whatever you see in NPL Code wiki web site.

The Console Page

Click `Menu::Tools::Console` to open `Console Window`. Here you can enter any NPL code or NPL web page code to be evalulated at runtime.

For example, enter following code and press `Run as Code` button:

```
print("hello world")
```

By default, text is output to `log.txt` file in the current working directory.

```
alert("hello world")
```

The above code will display a message box, if code wiki is launched with Paracraft.

The Log Page

Click `Menu::Tools::Log` to open `Log Window`. Both the console page and the log page can display content in `log.txt`.

- One can leave this window open even when your application restarts. It will automatically scroll to the newest log for you.
- Check `preserve log` to preserve logs between each application restarts.
- If there are critical errors or warnings, an error sign with a text number will be displayed in red on top of the log window, click it to navigate to the line of interest in the log file.

CHAPTER 13

Debugging And Logs

There are many ways to debug your NPL scripts, including printing logs, via [[NPL Code Wiki|NPLCodeWiki]], or via IDE like visual studio or vscode, etc.

Logs in NPL

`log.txt` file is the most important place to look for errors. When your application starts, NPL runtime will create a new `log.txt` in the current working directory.

Rule of Silence

The entire Unix system is built on top of the philosophy of rule of silence, which advocate every program to output nothing to standard output unless absolutely necessary.

So by default, NPL runtime output nothing to its standard output. All NPL output API like `log`, `print`, `echo` write content to `log.txt` instead of standard output. There is only one hidden API `io.write` which can be used to write to standard output.

By default, NPL runtime and its libraries output many info to log files. This log file can grow big if your server runs for days. There are two ways to deal with it: one is to use a timer (or external shell script) to backup or redirect `log.txt` to a different file everyday; second is to change the default log level to `error`, which only output fatal errors to `log.txt`. Application developers can also write their own logs to several files simultaneously, please see `script/ide/log.lua` for details.

When application starts, one can also change the default log file name, or whether it will overwrite existing `log.txt` or append to it. Please refer to log interface C++ API for details.

Use Logs For Debugging

The default log level of NPL is `TRACE`, which will write lots of logs to `log.txt`. At development time, it is good practice to keep this file open in a text editor that can automatically refresh when file content is changed.

When you feel something goes wrong, search for `error` or `runtime error` in `log.txt`. NPL Code Wiki provide a log page from `menu:view:log`, where you can filter logs. See below:

Many script programmers prefer to insert `echo({params, ...})` in code to output debug info to `log.txt`, and when code is working, these `echo` lines are commented out.

A good practice is to insert permanent log line with debug level set to `debug` like `LOG.std(nil, "debug", "modulename", "some text", ...)`.

Use A Graphical Debugger

HTTP Debugger in NPL Code Wiki

Sometimes inserting temporary log lines into code can be inefficient. One can also use NPL HTTP debugger to debug code via HTTP in a HTML5 web browser with nice GUI. NPL HTTP Debugger allows you to debug remote process, such as debugging linux/android/ios NPL process on a windows or mac computer. See [[NPLCodeWiki]] for details.

If you have installed NPL/Lua language service for visual studio, one can set HTTP-based breakpoints directly inside visual studio.

In visual studio, open your NPL script or page file, right click the line where you want to set breakpoint, and from the context menu, select `NPL Set Breakpoint Here`.

If you have started your NPL process, visual studio will open the HTTP debugger page with your default web browser. This is usually something like `http://localhost:8099/debugger`. Breakpoints will be automatically set and file opened in the browser.

Debugger Plugin for Visual Studio

We also provide open source NPL debugger plugins that is integrated with visual studio and vscode, see [[Install-Guide]] for details.However, it is a little complex to install visual studio debugger plugin, please follow instruction here: How To Install NPL Debugger for VS

Once installed, you can debug by selecting from visual studio menu `Tools::Launch NPL Debugger...`

CHAPTER 14

Embedding NPLRuntime

NPLRuntime comes with a general purpose executable for launching NPL based applications. This is the standard usage of NPLRuntime.

However, some client application may prefer embedding NPLRuntime in their own executable file, or even as child window of their own application. This is possible via NPLRuntime dll or (`paraengineclient.dll`) under windows platform.

Please see [[https://github.com/LiXizhi/ParaCraftSDK/blob/master/samples/MyAppExe/MyApp.cpp]] for details.

```
#include "PEtypes.h"
#include "IParaEngineApp.h"
#include "IParaEngineCore.h"
#include "INPL.h"
#include "INPLRuntime.h"
#include "INPLRuntimeState.h"
#include "INPLAcitvationFile.h"
#include "NPLInterface.hpp"
#include "PluginLoader.hpp"

#include "MyApp.h"

using namespace ParaEngine;
using namespace NPL;
using namespace MyCompany;

// some lines omitted here ....

INT WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR lpCmdLine, INT)
{
    std::string sAppCmdLine;
    if (lpCmdLine)
        sAppCmdLine = lpCmdLine;

    // TODO: add your custom command line here
    sAppCmdLine += " mc=true noupdate=true";
    CMyApp myApp(hInst);
    return myApp.Run(0, sAppCmdLine.c_str());
}
```

Building Custom Executable Application

To build your custom application, you must download NPLRuntime source code (for header files) and set CMAKE include directory properly.

- Download the sample at [[https://github.com/LiXizhi/ParaCraftSDK/blob/master/samples/MyAppExe]]
- Set CMAKE include directory to [[https://github.com/LiXizhi/NPLRuntime/tree/master/Client/trunk/ParaEngineClient/Core]]
- `paraengineclient.dll` is dynamically loaded at runtime via NPL plugin interface, which means that your executable does not need to link to any library file. **All you need when embedding NPLRuntime is header files. ** This decouples your host application executable from NPLRuntime library files which may be upgraded separately without rebuilding your host executable.
  - Main header files:
    - IParaEngineCore.h
    - IParaEngineApp.h

Build NPLRuntime as Libraries

In most cases, you do not need to build these libraries, simply copy `ParaEngineClient.dll` and other related dll files from the ParacraftSDK's `./redist` folder to your host application's executable directory. Make sure they are up to date.

However, one can also build NPL Runtime as Libraries manually. See `CMake` options when building NPLRuntime from source code.

CHAPTER 15

Extending NPLRuntime With C++ Plugins

NPL provides three extensibility modes: (1) NPL scripting (2) Mono C# dll (3) C++ plugin interface. All of them can be used simultaneously. Please see [[BasicConcept]] for details.

C++ plugins

C++ plugins allows us to treat dll/so file as a single script file. We would only want to use it for performance critical tasks or functions that make heavy use of other third-party C/C++ libraries.

The following is a sample c++ plugin. [[https://github.com/LiXizhi/ParaCraftSDK/tree/master/samples/plugins/HelloWorldCppDll]]

```
#include "stdafx.h"
#include "HelloWorld.h"

#ifdef WIN32
#define CORE_EXPORT_DECL    __declspec(dllexport)
#else
#define CORE_EXPORT_DECL
#endif

// forward declare of exported functions.
#ifdef __cplusplus
extern "C" {
#endif
    CORE_EXPORT_DECL const char* LibDescription();
    CORE_EXPORT_DECL int LibNumberClasses();
    CORE_EXPORT_DECL unsigned long LibVersion();
    CORE_EXPORT_DECL ParaEngine::ClassDescriptor* LibClassDesc(int i);
    CORE_EXPORT_DECL void LibInit();
    CORE_EXPORT_DECL void LibActivate(int nType, void* pVoid);
#ifdef __cplusplus
}   /* extern "C" */
#endif
```

```
HINSTANCE Instance = NULL;

using namespace ParaEngine;

ClassDescriptor* HelloWorldPlugin_GetClassDesc();
typedef ClassDescriptor* (*GetClassDescMethod)();

GetClassDescMethod Plugins[] =
{
    HelloWorldPlugin_GetClassDesc,
};

/** This has to be unique, change this id for each new plugin.
*/
#define HelloWorld_CLASS_ID Class_ID(0x2b905a29, 0x47b409af)

class HelloWorldPluginDesc:public ClassDescriptor
{
public:
    void* Create(bool loading = FALSE)
    {
        return new CHelloWorld();
    }

    const char* ClassName()
    {
        return "IHelloWorld";
    }

    SClass_ID SuperClassID()
    {
        return OBJECT_MODIFIER_CLASS_ID;
    }

    Class_ID ClassID()
    {
        return HelloWorld_CLASS_ID;
    }

    const char* Category()
    {
        return "HelloWorld";
    }

    const char* InternalName()
    {
        return "HelloWorld";
    }

    HINSTANCE HInstance()
    {
        extern HINSTANCE Instance;
        return Instance;
    }
};

ClassDescriptor* HelloWorldPlugin_GetClassDesc()
{
    static HelloWorldPluginDesc s_desc;
    return &s_desc;
}

CORE_EXPORT_DECL const char* LibDescription()
{
    return "ParaEngine HelloWorld Ver 1.0.0";
}

CORE_EXPORT_DECL unsigned long LibVersion()
{
    return 1;
}

CORE_EXPORT_DECL int LibNumberClasses()
{
    return sizeof(Plugins)/sizeof(Plugins[0]);
}

CORE_EXPORT_DECL ClassDescriptor* LibClassDesc(int i)
{
    if (i < LibNumberClasses() && Plugins[i])
    {
        return Plugins[i]();
    }
    else
    {
        return NULL;
    }
}

CORE_EXPORT_DECL void LibInit()
{
}

#ifdef WIN32
BOOL WINAPI DllMain(HINSTANCE hinstDLL,ULONG fdwReason,LPVOID lpvReserved)
#else
void __attribute__ ((constructor)) DllMain()
#endif
{
    // TODO: dll start up code here
#ifdef WIN32
    Instance = hinstDLL;                // Hang on to this DLL's instance handle.
    return (TRUE);
#endif
}

/** this is the main activate function to be called, when someone calls NPL.activate(
↪"this_file.dll", msg);
*/
CORE_EXPORT_DECL void LibActivate(int nType, void* pVoid)
{
    if(nType == ParaEngine::PluginActType_STATE)
    {
        NPL::INPLRuntimeState* pState = (NPL::INPLRuntimeState*)pVoid;
        const char* sMsg = pState->GetCurrentMsg();
        int nMsgLength = pState->GetCurrentMsgLength();
        NPLInterface::NPLObjectProxy input_msg =␣
↪NPLInterface::NPLHelper::MsgStringToNPLTable(sMsg);
        const std::string& sCmd = input_msg["cmd"];
        if(sCmd == "hello" || true)
        {
            NPLInterface::NPLObjectProxy output_msg;
            output_msg["succeed"] = "true";
            output_msg["sample_number_output"] = (double)(1234567);
            output_msg["result"] = "hello world!";

            std::string output;
            NPLInterface::NPLHelper::NPLTableToString("msg", output_msg, output);
            pState->activate("script/test/echo.lua", output.c_str(), output.size());
        }
    }
}
```

The most important function is `LibActivate` which is like `NPL.this(function() end)` in NPL script.

It is the function to be invoked when someone calls something like below:

```
NPL.activate("this_file.dll", {cmd="hello"})
```

One can also place dll or so file under sub directories of the working directory, such as `NPL.activate("mod/this_file.dll", {cmd="hello"})`

Note to Linux Developers

Under linux, the generated plugin file name is usually "libXXX.so", please copy this file to working directory and load using `NPL.activate("XXX.dll", {cmd="hello"})` instead of the actual filename. NPLRuntime will automatically replace `XXX.dll` with `libXXX.so` when searching for the dll file. This way, your script code does not need to change when deploying to linux and windows platform.

CHAPTER 16

NPL Basic Concept

This section provides a more detailed description to NPL's language feature.

A Brief History of Computer Language

Computer programming language is essentially a language in itself, like English and Chinese; but it is mainly written by humans to instruct computers. Our current computers has its unique way to process language. In general, it is a kind of command-like IO system, with input and output. Any piece of code can always be statically mapped to nodes and connections, where each connection has a single-unique direction from output to input; when a computer executes the code, it moves data along those connections. This is similar to how our brain works by our biological neural network. The difference is that we do not know how, when and where data is passed along. Since parallelism in our brain can also be mapped to a certain kind of node-connection relationship; the only thing differs is computational speed. So when designing computer language, let us first take concurrent programming out, and focus on making it easier and efficient to create all kinds of nodes and connections, and passing data along the connections.

Let us list all major patterns:

- Nearly all languages provides functions:
  - Functional language like `C` provides defining custom input/output patterns with functions. Let us regard `if,else,expressions`, etc as build-in functions.
- In some languages like `javascript`, function is first-class object, allowing one to create interesting node-connection patterns.
- Some like `C++, java, php` supports object-oriented programming, which is essentially a filter-like input/output patterns, that moves data as a whole more easily.
- Some is weakly typed like `javascript`, `python` allowing a node to accept input from more versatile input, at some expense. Some strongly-typed language like `C++` use template-programming to simulate this feature at the cost of more code generated.
- Some is statically scoped like `lua`, that automatically captured variables for any asynchronous callback, and some dynamically scoped language also simulates this feature with anonymous function and captures in some clumsy way like 'C#'.

- Some features establishing node and connections across multiple threads or computers on the network.

So what is the language that support all the above, while still maintaining high-performance? A close answer is `lua`; the complete answer is `NPL`.

What NPL is Not

- `meta programming` is a technique to translate text code from first domain to second one, and then compile second one and execute. It is common for some library to use this technique to provide a more concise way of coding with the library. Examples:
  - `qt.io` in C++
  - `php` which pre-process code-mixed hyper text into complete programming code.
  - `Scala` to 'Java'
  - `CoffeeScript` to `Javascript`
- NPL is NOT `meta programming`. But [[NPL web server|WebServer]] library use a similar technique to make coding server web pages that generate HTML5/javascript on the client more easily.
- NPL is not syntactic sugar or meta-programming for concurrent programming library. It is the programmer's job to write or use libraries that support concurrent programming, rather than introducing some non-intuitive syntax to the language itself.

NPL Runtime Architecture

This section helps you to understand the internals of NPL runtime. One of the major things that one need to understand is its message passing model.

Message Passing Model

In NPL, any file on disk can receive messages from other files. A file can send messages to other files via `NPL.activate` function. This function is strictly asynchronous. The details is below:

Message processing always has two phases:

- phase1: synapse data relay phase
- phase2: neuron file response phase

When you send a message via `NPL.activate`, the message is in phase 1. Message is not processed immediately, but cached in a queue. For example, if the target neuron file is on the local machine, the message is pushed (cached) directly to local thread queue; if the target is on remote machine, NPL runtime will automatcally establish TCP connection, and it will ensure the message is pushed(cached) on the target machine's specified thread queue.

Message processing takes place in phase2. Each NPL runtime process may contain one or more NPL runtime states, each corresponds to one system-level thread with its own message queue, memory, and code. NPL runtime states are logically separated from each other. In each NPL state, it processes messages in its message queue in heart-beat fashion. Therefore it is the programmer's job to finish processing the message within reasonable time slice, otherwise the message queue will be full and begin to lose early messages.

Code Overview

This section helps you to understand the open source code of NPL.

incoming connections

```
incoming connection:  acceptor/dispatcher thread
    -> CNPLNetServer::handle_accept
        -> CNPLNetServer.m_connection_manager.start(a new CNPLConnection object) : a␣
↪new CNPLConnection object is added to CNPLConnectionManager
            -> CNPLConnection.start()
                -> automatically assign a temporary nid to this unauthenticated␣
↪connection, temp connection is called ~XXX.
                    -> CNPLDispatcher::AddNPLConnection(temporary nid, CNPLConnection)
                -> begin reading the connection sockets
                -> LATER: Once authenticated, the scripting interface can call␣
↪(CNPLConnection)NPL.GetConnection(nid)::rename(nid, bAuthenticated)
```

if any messages are sent via the new incomming connection, the message field should contain msg.nid and msg.tid, where tid is temporaray nid. If there is no nid, it means that the connection is not yet authenticated. In NPL, one can call `NPL.accept(tid, nid)`

incoming messages

```
incoming message
    -> CNPLConnection::handle_read
        -> CNPLConnection::handleReceivedData
            -> CNPLConnection::m_parser.parse until a complete message is read
                -> CNPLConnection::handleMessageIn()
                    -> CNPLConnection::m_msg_dispatcher.DispatchMsg((NPLMsgIn)this->m_
↪input_msg)
                        -> CNPLRuntimeState = GetNPLRuntime()->
↪GetRuntimeState(NPLMsgIn::m_rts_name)
                        -> if CNPLDispatcher::CheckPubFile(NPLMsgIn::m_filename)
                            -> new NPLMessage(from NPLMsgIn)
                            -> CNPLRuntimeState::Activate_async(NPLMessage);
                                -> CNPLRuntimeState::SendMessage(NPLMessage) ->␣
↪insert message to CNPLRuntimeState.m_input_queue
                        -> or return access denied
```

outgoing messages

```
from NPL script NPL.activate()
    -> CNPLRuntime::NPL_Activate
        -> if local activation, get the runtime state.
            -> CNPLRuntimeState::Activate_async(NPLMessage) ->␣
↪CNPLRuntimeState::SendMessage(NPLMessage) -> insert message to CNPLRuntimeState.m_
↪input_queue
        -> if remote activation with nid, send via dispatcher
            -> CNPLDispatcher::Activate_Async(NPLFileName, code)
                -> CNPLConnection = CNPLDispatcher::CreateGetNPLConnectionByNID(nid)
                    -> if found in CNPLDispatcher::m_active_connection_map(nid,␣
↪CNPLConnection), return
```

```
                    -> or if found in CNPLDispatcher::m_pending_connection_map(nid,␣
↪CNPLConnection), return
                    -> or if found in CNPLDispatcher::m_server_address_map(host, port,
↪ nid): we will actively establish a new connection to it.
                        -> CNPLDispatcher::CreateConnection(NPLRuntimeAddress)
                            -> CNPLNetServer::CreateConnection(NPLRuntimeAddress)
                                -> new CNPLConnection()
                                ->␣
↪CNPLConnection::SetNPLRuntimeAddress(NPLRuntimeAddress)
                                -> (CNPLNetServer::m_connection_manager as␣
↪CNPLConnectionManager)::add(CNPLConnection)
                                -> CNPLConnection::connect() -> CNPLConnection::m_
↪resolver->async_resolve ->
                                    -> CNPLConnection::handle_resolve ->␣
↪CNPLConnection::m_socket.async_connect
                                    -> CNPLConnection::handle_connect
                                        -> CNPLConnection::handleConnect() : for␣
↪custom behavior
                                        -> CNPLConnection::start()
                                            ->␣
↪CNPLDispatcher::AddNPLConnection(CNPLConnection.m_address.nid, CNPLConnection)
                                                -> add nid to CNPLDispatcher::m_
↪active_connection_map(nid, CNPLConnection), remove nid from m_pending_connection_
↪map(nid, CNPLConnection)
                                            -> begin reading the connection sockets
                            -> Add to CNPLDispatcher::m_pending_connection_map if not␣
↪connected when returned, return the newly created connection
                    -> or return null.
                -> CNPLConnection::SendMessage(NPLFilename, code)
                    -> new NPLMsgOut( from NPLFilename, code)
                    -> push to CNPLConnection::m_queueOutput() and start sending␣
↪first one if queue is previously empty.
```

NPL Message Format

NPL message protocol is based on TCP and HTTP compatible. More information is in NPLMsgIn_parser.h

quick sample

```
A (g1)script/hello.lua NPL/1.0
rts:r1
User-Agent:NPL
16:{"hello world!"}
```

NPL Quick Guide

In NPL runtime, there can be one or more runtime threads called NPL runtime states. Each runtime state has its own name, stack, thread local memory manager, and message queue. The default NPL state is called (main), which is also the thread for rendering 3D graphics. It is also the thread to spawn other worker threads.

To make a NPL runtime accessible by other NPL runtimes, one must call

```
    NPL.StartNetServer("127.0.0.1", "60001");
```

This must be done on both client and server. Please note that NPL does not distinguish between client and server. If you like, you may call the one who first calls `NPL.activate` to be client. For private client, the port number can be "0" to indicate that it will not be accessible by other computers, but it can always connect to other public NPL runtimes.

To make files within NPL runtime accessible by other NPL runtimes, one must add those files to public file lists by calling

```
    NPL.AddPublicFile("script/test/network/TestSimpleClient.lua", 1);
    NPL.AddPublicFile("script/test/network/TestSimpleServer.lua", 2);
```

the second parameter is id of the file. The file name to id map must be the same for all communicating computers. This presumption may be removed in futhure versions.

To create additional worker threads (NPL states) for handling messages, one can call

```
    for i=1, 10 do
        local worker = NPL.CreateRuntimeState("some_runtime_state"..i, 0);
        worker:Start();
    end
```

Each NPL runtime on the network (either LAN/WAN) is identified by a globally unique `nid` string. NPL itself does not verify or authenticate nid, it is the programmers' job to do it.

To add a trusted server address to nid map, one can call

```
    NPL.AddNPLRuntimeAddress({host="127.0.0.1", port="60001", nid="this_is_server_nid
↪"})
```

so that NPL runtime knows how to connection to the nid via TCP endpoints. host can be ip or domain name.

To activate a file on a remote NPL runtime(identified by a nid), one can call

```
while( NPL.activate("(some_runtime_state)this_is_server_nid:script/test/network/
↪TestSimpleServer.lua",
 {some_data_table}) ~=0 ) do
end
```

Please note that `NPL.activate()` function will return non-zero if no connection is established for the target, but it will immediately try to establish a new connection if the target address of server nid is known. The above sample code simply loop until activate succeed; in real world, one should use a timer with timeout or use one of the helper function such as `NPL.activate_with_timeout`.

When a file is activated, its activation function will be called, and data is inside the global msg table.

```
    local function activate()
        if(msg.nid) then
        elseif(msg.tid) then
        end
        NPL.activate(string.format("%s:script/test/network/TestSimpleClient.lua", msg.
↪nid or msg.tid), {some_date_sent_back})
    end
    NPL.this(activate)
```

`msg.nid` contains the nid of the source NPL runtime. `msg.nid` is nil, if connection is not authenticated or nid is not known. In such cases, `msg.tid` is always available; it contains the local temporary id. When the connection is authenticated, one can call following function to reassign nid to a connection or reject it. NPL runtime will try to reuse the same existing TCP connect between current computer and each nid or tid target, for all communcations between the two endpoints in either direction.

```
    NPL.accept(msg.tid, nid)  -- to rename tid to nid, or
    NPL.reject(msg.tid) -- to close the connection.
```

only the main thread support non-thread safe functions, such as those with rendering. For worker thread(runtime state), please only use NPL functions that are explicitly marked as thread-safe in the documentation.

NPL Extensibility and Scripting Performance

NPL provides three extensibility modes: (1) Lua scripting runtime states (2) Mono .Net runtimes (3) C++ plugin interface All of them can be used simultanously to create logics for game applications. All modes support cross-platform and multi-threaded computing.

Lua runtime is natively supported and has the most extensive ParaEngine API exposed. It is both extremely light weighted and having a good thread local memory manager. It is suitable for both client and server side scripting. It is recommended to use only lua scripts on the client side.

Mono .Net runtimes is supported via NPLMono.dll (which in turn is a C++ plugin dll). The main advantage of using .Net scripting is its rich libraries, popular language support(C#,Java,VB and their IDE) and compiled binary code performance. .Net scripting runtime is recommended to use on the server side only, since deploying .Net on the client requires quite some disk space. And .Net (strong typed language) is less effective than lua when coding for client logics.

C++ plugins allows us to treat dll file as a single script file. It has the best performance among others, but also is the most difficult to code. We would only want to use it for performance critical tasks or functions that make extensive use of other third-party C/C++ libraries. For example, NPLRouter.dll is a C++ plugin for routing messages on the server side.

Cross-Runtime Communication

A game project may have thousands of lua scripts on the client, mega bytes of C# code contained in several .NET dll assemblies on the server, and a few C++ plugin dlls on both client and server. (On linux, *.dll will actually find the *.so file) All these extensible codes are connected to the ParaEngine Runtime and have full access to the exposed ParaEngine API. Hence, source code written in different language runtimes can use the ParaEngine API to communicate with each other as well as the core game engine module.

More over, `NPL.activate` is the single most versatile and effective communication function used in all NPL extensibility modes. In NPL, a file is the smallest communication entity. Each file can receive messages either from the local or remote files.

- In NPL lua runtime, each script file with a activate() function assigned can receive messages from other scripts.
- In NPL .Net runtime, each C# file with a activate() function defined inside a class having the same name as the file name can receive messages from other scripts. (namespace around class name is also supported, see the example)
- In NPL C++ plugins, each dll file must expose a activate() function to receive messages from other scripts.

Following are example scripts and NPL.activate() functions to send messages to them.

```
----------------------------------------
File name:  script/HelloWorld.lua  (NPL script file)
activate:   NPL.activate("script/HelloWorld.lua", {data})
----------------------------------------
local function activate()
end
```

```
NPL.this(activate)

---------------------------------------
File name:  HelloWorld.cs inside C# mono/MyMonoLib.dll
activate:   NPL.activate("mono/MyMonoLib.dll/HelloWorld.cs", {data})
            NPL.activate("mono/MyMonoLib.dll/ParaMono.HelloWorld.cs", {data})
---------------------------------------
class HelloWorld
{
    public static void activate(ref int type, ref string msg)
    {
    }
}
namespace ParaMono
{
    class HelloWorld
    {
        public static void activate(ref int type, ref string msg)
        {
        }
    }
}
---------------------------------------
File name:  HelloWorld.cpp inside C++ MyPlugin.dll
activate:   NPL.activate("MyPlugin.dll", {data})
---------------------------------------
CORE_EXPORT_DECL void LibActivate(int nType, void* pVoid)
{
    if(nType == ParaEngine::PluginActType_STATE)
    {
    }
}
```

The `NPL.activate()` can use the file extension (*.lua, *.cs, *.dll) to distinguish the file type.

All `NPL.activate()` communications are asynchronous and each message must go through message queues between the sending and receiving endpoints. The performance of NPL.activate() is therefore not as efficient as local function calls, so they are generally only used to dispatch work tasks to different worker threads or communicating between client and server.Generally, 30000 msgs per second can be processed for 100 bytes message on 2.5G quad core CPU machine. For pure client side game logics, we rarely use NPL.activate(), instead we use local function calls whereever possible.

High Level Networking Programming Libraries

In real world applications, we usually need to build a network architecture on top of NPL, which usually contains the following tool sets and libraries.

- gateway servers and game(application) servers where the client connections are kept.
- routers for dispaching messagings between game(app) servers and database servers.
- database servers that performances business logics and query the underlying databases(My SQLs)
- memory cache servers that cache data for database servers to minimize access to database.
- Configuration files and management tools (shell scripts, server deployingment tools, monitoring tools, etc)
- backup servers for all above and DNS servers.

CHAPTER 17

Bootstrapping

Bootstrapping is to specify the first script file to load when starting NPL runtime, it will also be activated every 0.5 seconds. There are several ways to do it via [[command line|NPLCommandLine]].

The recommended way is to specify it explicitly with `bootstrapper=filename` parameters. Like below.

```
npl bootstrapper="script/gameinterface.lua"
```

If no file name is specified, `script/gameinterface.lua` will be used. A more concise way is to specify the file name as first parameter, like below.

```
npl helloworld.lua
```

Internally, the boot order of NPL is like below:

- When the NPL runtime starts, it first reads all of its command line arguments as name, value pairs and save them to an internal data structure for later use.
- NPL recognizes several predefined command names, one of them is bootstrapper. It specifies the main loop file.
- NPL initializes itself. This can take several seconds, involving reading configurations, graphics initialization, script API bindings, etc.
- When everything is initialized, the majority of NPL's Core API is available to the scripting interface and NPL loads the main loop script specified by the bootstrapper and calls its activation function every 0.5 seconds.
- Anything happens afterwards is up to the progammer who wrote the main loop script.

In addition to script file, one can also use XML file as the bootstrapper. The content of the xml should look like below.

```
<?xml version="1.0" ?>
<MainGameLoop>(gl)script/apps/Taurus/main_loop.lua</MainGameLoop>
```

To start it, we can use

```
npl script/apps/Taurus/bootstrapper.xml
```

Understanding File Path

When programming with NPL, the code will reference other script or asset file by relative file name. The search order of a file is as follows:

- check if file exists in the current development directory (in release time, this is the same as the working directory)
- check if file exists in the search paths (by default it is the current working directory)
- check if file exists in any loaded archive files (mainXXX.PKG or world/plugins ZIP files) in their loading order.
- if the file is NPL script file, look for precompiled binary file at ./bin/*filename*.o also in above places and order.
- if the file is listed in `asset_manifest.txt`, we will download it or open it from `./temp/cache` if already downloaded.
- check if disk file exists in all search paths of [[npl_packages]].
- if none of above places are found, we will report file not found. Please also note, some API allows you to use specify search order, temporarily disable some places, or include certain directory like the current writable directory. Please refer to the usage of each API for details.

CHAPTER 18

Build-in NPL CommandLine

Native Params

The following command line is built-in to NPL executable.

- `bootstrapper="myapp/main.lua"`: set bootstrapper file
- [-d|D] [filename]: -d to run in background (daemon mode), -D to force run in fore ground.
- `dev="C:/MyDev/"`: set dev directory, if empty or "/", it means the current working directory.
- `servermode="true"`: disable 3D rendering and force running in server mode.
- `logfile="log2016.5.20.txt"`: change default `log.txt` location
- `single="true"`: force only one instance of the executable can be running in the OS.
- `loadpackage="folder1,folder1,..."`: commar separated list of packages to load before bootstrapping. Such as `loadpackage="npl_packages/paracraft/"`

NPL System Module Params

The following params are handled by the NPL system module

- `debug="main"` : enable IPC debugging to the main NPL thread. No performance penalties.
- `resolution="1020 680"`: force window size in case of client application

Paracraft Module Params

The following params are handled by the Paracraft Package

- `world="worlds/DesignHouse/myworld"` : Load a given world at start up.

- `mc="true"`: force paracraft (in case of running from MagicHaqi directory)

CHAPTER 19

NPL Load File

In NPL code, one may call something like `NPL.load("script/ide/commonlib.lua")` to load a file. All NPL code needs to be compiled in order to run. `NPL.load` does following things automatically for you.

- find the correct file either in NPL zip package or according to NPL search path defined in [[npl_packages]] and automatically use precompiled binary version (`*.o`) if available (see [[DeployGuide]]).
  - relative path is supported, but only recommended for private files, such as `NPL.load("./A.lua") or NPL.load("../../B.lua")`
  - when using relative path, file extension can be omitted, where *.npl and *.lua are searched, such as `NPL.load("./A") or NPL.load("../../B")`
- automatically resolve recursion.(such as File A load File B, while B also load A).
- compiles the script code if not yet compiled
  - if file extension is `*.npl`, NPL meta compiler will be invoked.
- The first time code is compiled, it also execute the code chunk in protected mode with `pcall`. In most cases, this means injecting new code to the global or exported table, so that one can use them later.
- it can also be used to load `C++/Mono C#/NPL packages`
- return the exported file module if any. See `file based module` section below for details.

```
/**
load a new file (in the current runtime state) without activating it. If the file is␣
↪already loaded,
it will not be loaded again unless bReload is true.
IMPORTANT: this function is synchronous; unlike the asynchronous activation function.
LoadFile is more like "include in C++".When the function returns, contents in the␣
↪file is loaded to memory.
@note: in NPL/lua, function is first class object, so loading a file means executing␣
↪the code chunk in protected mode with pcall,
in most cases,  this means injecting new code to the global table. Since there may be␣
↪recursions (such as A load B, while B also load A),
your loading code should not reply on the loading order to work. You need to follow␣
↪basic code injection principles.
```

```
For example, commonlib.gettable("") is the the commended way to inject new code to␣
↪the current thread's global table.
Be careful not to pollute the global table too much, use nested table/namespace.
Different NPL applications may have their own sandbox environments, which have their␣
↪own dedicated global tables, for example all `*.page` files use a separate global␣
↪table per URL request in NPL Web Server App.
@note: when loading an NPL file, we will first find if there is an up to date␣
↪compiled version in the bin directory. if there is,
we will load the compiled version, otherwise we will use the text version.  use bin␣
↪version, if source version does not exist; use bin version, if source and bin␣
↪versions are both on disk (instead of zip) and that bin version is newer than the␣
↪source version.
e.g. we can compile source to bin directory with file extension ".o", e.g. "script/
↪abc.lua" can be compiled to "bin/script/abc.o", The latter will be used if␣
↪available and up-to-date.
@param filePath: the local relative file path.
If the file extension is ".dll", it will be treated as a plug-in. Examples:
    "NPLRouter.dll"         -- load a C++ or C# dll. Please note that, in windows, it␣
↪looks for NPLRonter.dll; in linux, it looks for ./libNPLRouter.so
    "plugin/libNPLRouter.dll"           -- almost same as above, it is reformatted to␣
↪remove the heading 'lib' when loading. In windows, it looks for plugin/NPLRonter.
↪dll; in linux, it looks for ./plugin/libNPLRouter.so
@param bReload: if true, the file will be reloaded even if it is already loaded.
   otherwise, the file will only be loaded if it is not loaded yet.
@remark: one should be very careful when calling with bReload set to true, since this␣
↪may lead to recursive
    reloading of the same file. If this occurs, it will generate C Stack overflow␣
↪error message.
*/
NPL.load(filePath, bReload)
```

Code Injection Model

Since, in NPL/lua, table and functions are first class object, we use a very flexible code injection model to manage all dynamically loaded code during the life time of an application.

To inject new code, we use method like `commonlib.gettable`, `commonlib.inherit`, please see [[ObjectOriented]] for details. The developer should ensure they inject their code with unique names (such as `CompanyName.AppName.ModuleName` etc). In other words, do not pollute the global table.

To reference these code, the user needs to call `NPL.load` as well as `commonlib.gettable` to create a local stub variable in the file scope. Please note, due to `NPL.load` order, the stub may be created before the actual code is injected into it.

For example, you have a class file in `script/MyApp/MyClass.lua` like below

```lua
local MyClass = commonlib.inherit(nil, commonlib.gettable("MyApp.MyClass"));

MyClass.default_param = 1;

-- this is the constructor function.
function MyClass:ctor()
   self.map = {};
end

function MyClass:init(param1)
```

```
   self.map.param1 = param1;
   return self;
end

function MyClass:Clone()
   return MyClass:new():init(self:GetParam());
end

function MyClass:GetParam()
   return self.map.param1;
end
```

To use above class, one may use

```
NPL.load("(gl)script/MyApp/MyClass.lua")
local MyClass = commonlib.gettable("MyApp.MyClass");

local UserClass = commonlib.inherit(nil, commonlib.gettable("MyApp.UserClass"));

function UserClass:ctor()
   self.map = MyClass:new();  -- use another class
end
```

please see [[ObjectOriented]] for details.

File-based Modules

File-based modules use the containing source code filename to store exported object. So programmers do not need to explicitly specify object namespace in code, this solves several issues:

- code written in this style is independent of its containing file location.
- code looks more isolated.
- multiple versions of the same code can coexist.

  One not-so-convenient thing is that the user must explicitly load every referenced external modules written in this way in every file. This is the primary reason why the common NPL library are NOT written in this way.

Defining file-based module

There are three ways to define a file-based module. Suppose, you have a file called `A.npl` and you want to export some object from it.

- One way is to call `NPL.export` when file is loaded.

```
-- this is A.npl file
local A = {};
NPL.export(A);

function A.print(s)
  echo(s);
end
```

Following is a better and recommended way, which works with cyclic dependencies. See `Module Cyclic Reference` section for details.

```
-- this is A.npl file
local A = NPL.export();
function A.print(s)
  echo(s);
end
```

- Another way is simply return the object to be associated with the current file at the last line of code, like below. This is also a lua-compatible way. It does not work when there are cyclic dependencies.

```
-- this is A.npl file
local A = {};
return A;
```

- finally, there is an advanced manual way to simply add to the `_file_mod_` table.

```
-- this is A.npl file
local A = {};
_file_mod_[NPL.filename():gsub("([^/]*$)", "").."A.npl"] = A;
```

As you can see, file-based module simply automatically stores the mapping from the module's full filename to its exported object in a hidden global table. If one changes the filename or file location, the mapping key also changes. This is why you can load multiple versions of the same module and let the user to choose which one to use in a given source file.

Module Cyclic Reference

When you write file modules that depends on each file, there are cyclic dependencies. One workaround is to modify the default exported object, which is always an empty table, instead of creating a local table. See below. We have two files A and B that reference each other.

```
-- A.lua
local B = NPL.load("./B.lua")
local A = NPL.export();
function A.print(s)
  B.print(s)
end
```

```
-- B.lua
local A = NPL.load("./A.lua")
local B = NPL.export();
function B.print(s)
  echo(s..type(A));
end
```

`NPL.export()` is the core of file module system in NPL, it will return the default empty table representing the current file. This is the same way as how `commonlib.gettable()` solve cyclic dependency.

Modules with Object Oriented Class Inheritance

The following shows an example of two class table with inheritance, and also cyclic dependency.

```
-- base_class.lua
local derived_class = NPL.load("./derived_class.lua")
local base_class = commonlib.inherit(nil, NPL.export());
function base_class:ctor()
   derived_class:cyclic_dependency_test();
end
```

```
-- derived_class.lua
local base_class = NPL.load("./base_class.lua")
local B = commonlib.inherit(base_class, NPL.export());
function B:ctor()
end
function B:cyclic_dependency_test()
end
```

Loading file-based module With module name

To import a file-based module, one calls `NPL.load(modname)`. `NPL.load` is a versatile function which can load either standard source files or file-based modules and return the exported module object.

> A `modname` is different from filename, a string is treated as `modname` if and only if it does NOT contain file extension or begin with "./" or "../". such as `NPL.load("sample_mod")`

`NPL.load(modname)` will automatically find the source file of the module by trying the following locations in order until it finds one. Using module name in NPL.load is less efficient than using explicit filename, because it needs to do the search every time it is called, so only use it at file-load time, such as at the beginning of a file.

- if the code that invokes `NPL.load` is from `npl_packages/[parentModDir]/`
  - try: `npl_packages/[parentModDir]/npl_mod/[modname]/[modname].lua|npl`
  - try: `npl_packages/[parentModDir]/npl_packages/[modname]/npl_mod/[modname]/[modname].lua|npl`
- try: `npl_mod/[modname]/[modname].npl|lua`
- try: `npl_packages/[modname]/npl_mod/[modname]/[modname].npl|lua`

Example 1: NPL.load("sample_mod") will test following file locations for both *.npl and *.lua file:

- if code that invokes it is from `npl_packages/[parentModDir]/`
  - `npl_packages/[parentModDir]/npl_mod/sample_mod/sample_mod.npl`
  - `npl_packages/[parentModDir]/npl_packages/sample_mod/npl_mod/sample_mod/sample_mod.npl`
- `npl_mod/sample_mod/sample_mod.npl`
- `npl_packages/sample_mod/npl_mod/sample_mod/sample_mod.npl`

Example 2: NPL.load("sample_mod.xxx.yyy") will test following file locations for both *.npl and *.lua file:

- if code that invokes it is from `npl_packages/[parentModDir]/`
  - `npl_packages/[parentModDir]/npl_mod/sample_mod/xxx/yyy.npl`
  - `npl_packages/[parentModDir]/npl_packages/sample_mod/npl_mod/sample_mod/xxx/yyy.npl`
- `npl_mod/sample_mod/xxx/yyy.npl`

- `npl_packages/sample_mod/npl_mod/sample_mod/xxx/yyy.npl`

As you can see, the `npl_packages/` and `npl_mod/` are two special folders to search for when loading file based module. code inside npl_packages/xxx/ folder always use local `npl_mod` and local npl_packages before searching for global ones, this allow multiple versions of the same npl_mod to coexist in different npl_packages. It is up to the developer to decide how to package and release their application or modules.

Here is an example of npl_mod in main package

For more information on NPL packages, please see [[npl_packages]]

When to Use `require`

`require` is Lua's way to load a file based module. NPL hooks into this function to always call `NPL.load` with the same input before passing down. More specifically, NPL injects a custom loader (at index 2) into the lua's `package.loaders`. So calling `require` is almost identical to calling `NPL.load` so far, except that multiple code versions can not coexist.

Do not Use `require`

Since NPL.load now hooks original lua's require function. Facts in this section may not be true, but we still do not recommend using `require` in your own code, unless you are reusing third-party lua libraries. The reason is simple, NPL.load is backward-compatible with the original require, but the original require does not support features provided by NPL.load. Using require in your own code may confuse other lua developers. Therefore, code written for NPL developers should always use NPL.load.

One should only use `require` to load lua C plugins, do not use it in your own NPL scripts. `require` is rated low in community. Its original implementation is same, but use a similar global hidden table, whose logic you never know, in a flat manner. It also does not support cyclic dependency so far.

Moreover, some application may need to create sandbox environment to isolate code execution (code are injected to specified global table), using `require` give you no control of code injection.

Different NPL applications may have their own sandbox environments, which have their own dedicated global tables, for example all `*.page` files use a separate global table per URL request in NPL Web Server App.

Finally, require' is slow and uses different search paths than NPL; while NPL.load is fast and honors precompiled code either in package zip file or in search paths defined in [[npl_packages]], and consistent on all platforms.

CHAPTER 20

File Activation

NPL can communicate with remote NPL script using the `NPL.activate` function. It is a powerful function, and is available in C++, mono plugin too.

Basic Syntax

Like in neural network, all communications are asynchronous and uni-directional without callbacks. Although the function does return a integer value. Please see `NPL Reference` for details.

```
NPL.activate(url, {any_pure_data_table_here, })
```

- @param `url`: a globally unique name of a NPL file name instance. The string format of an NPL file name is like below. `[(sRuntimeStateName|gl)][sNID:]sRelativePath[]` the following is a list of all valid file name combinations:
  - `user001@paraengine.com:script/hello.lua` – a file of user001 in its default gaming thread
  - `(world1)server001@paraengine.com:script/hello.lua` – a file of server001 in its thread world1
  - `(worker1)script/hello.lua` – a local file in the thread worker1
  - `(gl)script/hello.lua` – a glia (local) file in the current runtime state's thread
  - `script/hello.lua` – a file in the current thread. For a single threaded application, this is usually enough.
  - `(worker1)NPLRouter.dll` – activate a C++ file. Please note that, in windows, it looks for NPLRonter.dll; in linux, it looks for ./libNPLRouter.so
  - `(worker1)DBServer.dll/DBServer.DBServer.cs` – for C# file, the class must have a static activate function defined in the CS file.
- @param `msg`: it is a chunk of pure data table that will be transmitted to the destination file.

Neuron File

Only files associated with an activate function can be activated. This is done differently in NPL/C++/C# plugin

- In NPL, it is most flexible by using the build-in `NPL.this` function

```
local function activate()
   -- input is passed in a global "msg" variable
   echo(msg);
end
NPL.this(activate);
```

- `msg` is passed in a global `msg` variable which is visible to all files, and the `msg` variable will last until the thread receives the next activation message.
- In NPL's C++ plugin, you need to define a `C` function. Please see ParacraftSDK's example folder for details.
- In NPL's mono C# plugin, you simply define a class with a static `activate` function. Please see ParacraftSDK's example folder for details.

Input Message, `msg.nid` and `msg.tid`

In above code, `msg` contains the information received from the sender, plus the `source id` of the sender. For unauthenticated senders, the source id is stored in `msg.tid`, which is an auto-generated number string like "~1". The receiver can keep using this temporary id `msg.tid` to send message back, such as

```
local function activate()
   -- input is passed in a global "msg" variable
   NPL.activate(format("%s:some_reply_file.lua", msg.tid or msg.nid), {"replied"});
end
NPL.this(activate);
```

The receiver can also rename the temporary `msg.tid` by calling `NPL.accept(msg.tid, nid_name)`, so the next time if the receiver got a message from the same sender (i.e. the same TCP connection), the `msg.nid` contains the last assigned name and `msg.tid` field no longer exists. We usually use `NPL.accept` to distinguish between authenticated and unauthenticated senders, and reject unauthenticated messages by calling `NPL.reject(msg.tid)` as early as possible to save CPU cycles.

See following example:

```
local function activate()
   -- input is passed in a global "msg" variable
   if(msg.tid) then
      -- unauthenticated? reject as early as possible or accept it.
      if(msg.password=="123") then
         NPL.accept(msg.tid, msg.username or "default_user");
      else
         NPL.reject(msg.tid);
      end
   elseif(msg.nid) then
      -- only respond to authenticated messages.
      NPL.activate(format("%s:some_reply_file.lua", msg.nid), {"replied"});
   end
end
NPL.this(activate);
```

Please note, `msg.tid or msg.nid` always match to a single low-level TCP connection, hence their names are shared to all neuron files in the process. For example, if you accept in one neuron file, all other neuron files will receive messages in `msg.nid` form.

Please see

Neuron File Visibility

For security reasons, all neuron files can be activated by other files in the same process. This includes scripts in other local threads of the same process. See also [[MultiThreading]].

To expose script to remote computers, one needs to do two things.

- one is to start NPL server by listening to an IP and port: NPL uses TCP protocol for all communications.
- second is to tell NPL runtime that a given file is a `public neuron file`.

See below:

```
NPL.StartNetServer("0.0.0.0", 8080);
NPL.AddPublicFile(filename, id);
```

- "0.0.0.0" means all IP addresses, one can also use "127.0.0.1", "localhost" or whatever IP addresses.
- 8080: is the port number. Pick any one you like.

The second parameter to `NPL.AddPublicFile` is an integer, which is transmitted on behalf of the long filename to save bandwidth. So it must be unique if you add multiple public files.

> Please note, that file name must be relative to working directory. such as `NPL.AddPublicFile("script/test/test.lua", 1)`. Absolute path is not supported at the moment.

Activating remote file

Once a server NPL runtime exposes a public file, other client NPL runtime can activate it with the `NPL.activate` function. Please note that, an NPL runtime can be both server and client. The one who started the connection is usually called server. Pure client must also call `NPL.StartNetServer` in order to activate the server. But it can specify `port="0"` to indicate that it will not listen for incoming connections.

However, on the client, we need to assign a local name to the remote server using the `NPL.AddNPLRuntimeAddress`, so that we can refer to this server by name in all subsequent `NPL.activate` call.

```
NPL.AddNPLRuntimeAddress({host = "127.0.0.1", port = "8099", nid = "server1"})
```

Usually we do this during initialization time only once. After that we can activate public files on the remote server like below:

```
NPL.activate("server1:helloworld.lua", {})
```

Please note the name specified by `nid` is arbitrary and used only on the client computer to refer to a computer. In other words, different clients can name the same remote computer differently.

Message Delivery Guarantees

Please note that the first time that a computer activate a remote file, a TCP connection is automatically established, but the first message is NOT delivered. This is because `NPL.activate()` is asynchronous, it must return a value before the new connection is established. It always returns 0 when your message is delivered via an existing path, and non-zero in case of first message to a remote system.

If there is no already established path (i.e no TCP connection), NPL will immediately try to establish it. However, please note, the message that returns non-zero is NOT delivered, even if NPL successfully established a path to the remote system soon afterwards. Thus it is the programmer's job to activate again until `NPL.activate` returns 0. This guarantees that a message with 0 return value is always delivered at least in the viewpoint of NPL runtime.

The same mechanism can be used to recover lost-connections.

To write fault-tolerant message passing code, consider following.

- When a NPL runtime process start, ping remote process with NPL.activate until it returns 0 to establish TCP connections. This ensures that all subsequent NPL.activate across these two systems can be delivered.
- Discover Lost Connections:
  - Method1: Use a timer to ping or listen for disconnect system event and reconnect in case connection is lost. However, individual messages may be lost during these time.
  - Method2: Use a wrapper function to call NPL.activate, which checks its return value. If it is non-zero, either reconnect with timeout or put message to a pending queue in case connection can be recovered shortly and resend queued messages.

We leave it to the programmer to handle all occasions when NPL.activate returns non-zero values, since different business logic may use a different approach.

- Akka Reference

Example Client/Server app

To run the example, call following.

```
npl "script/test/network/SimpleClientServer.lua" server="true"
npl "script/test/network/SimpleClientServer.lua" client="true"
```

The source code of this demo is also included in `ParaCraftSDK/examples` folder.

filename: `script/test/network/SimpleClientServer.lua`

```
--[[
Author: Li,Xizhi
Date: 2009-6-29
Desc: start one server, and at least one client.
-----------------------------------------------
npl "script/test/network/SimpleClientServer.lua" server="true"
npl "script/test/network/SimpleClientServer.lua" client="true"
-----------------------------------------------
]]
NPL.load("(gl)script/ide/commonlib.lua"); -- many sub dependency included

local nServerThreadCount = 2;
local initialized;
local isServerInstance = ParaEngine.GetAppCommandLineByParam("server","false") ==
↪"true";
```

```
-- expose these files. client/server usually share the same public files
local function AddPublicFiles()
    NPL.AddPublicFile("script/test/network/SimpleClientServer.lua", 1);
end

-- NPL simple server
local function InitServer()
    AddPublicFiles();

    NPL.StartNetServer("127.0.0.1", "60001");

    for i=1, nServerThreadCount do
        local rts_name = "worker"..i;
        local worker = NPL.CreateRuntimeState(rts_name, 0);
        worker:Start();
    end

    LOG.std(nil, "info", "Server", "server is started with %d threads",␣
↪nServerThreadCount);
end

-- NPL simple client
local function InitClient()
    AddPublicFiles();

    -- since this is a pure client, no need to listen to any port.
    NPL.StartNetServer("0", "0");

    -- add the server address
    NPL.AddNPLRuntimeAddress({host="127.0.0.1", port="60001", nid="simpleserver"})

    LOG.std(nil, "info", "Client", "started");

    -- activate a remote neuron file on each thread on the server
    for i=1, nServerThreadCount do
        local rts_name = "worker"..i;
        while( NPL.activate(string.format("(%s)simpleserver:script/test/network/
↪SimpleClientServer.lua", rts_name),
            {TestCase = "TP", data="from client"}) ~=0 ) do
            -- if can not send message, try again.
            echo("failed to send message");
            ParaEngine.Sleep(1);
        end
    end
end

local function activate()
    if(not initialized) then
        initialized = true;
        if(isServerInstance) then
            InitServer();
        else
            InitClient();
        end
    elseif(msg and msg.TestCase) then
        LOG.std(nil, "info", "test", "%s got a message", isServerInstance and "server
↪" or "client");
        echo(msg);
    end
end
NPL.this(activate);
```

The above server is actually multi-threaded, please see [[MultiThreading]] for details.

> The first time, `NPL.activate` calls a new remote server (with which we have not established TCP connection), the message is dropped and returned a non-zero value. `NPLExtension.lua` contains a number of helper functions to help you for sending a guaranteed message, such as `NPL.activate_with_timeout`.You need to include [[commonlib]] to use it.

Trusted Connections and NID

In the receiver's activate function, it can assign any name or `nid` to incoming connection's NPL runtime.

HelloWorld Example

Now, here comes a more complicated helloworld. It turns an ordinary `helloworld.lua` file into a neuron file, by associating an `activate` function with it. The file is then callable from any NPL thread or remote computer by its NPL address(url).

```
local function activate()
   if(msg) then
      print(msg.data or "");
   end
   NPL.activate("(gl)helloworld.lua", {data="hello world!"})
end
NPL.this(activate);
```

Simple Web Server Example

NPL uses a HTTP-compatible protocol, so it is possible to handle standard HTTP request using the same NPL server. When NPL runtime receives a HTTP request message, it will send it to a publicly visible file with id `-10`. So we can create a simple HTTP web server with just a number of lines, like below:

filename: `main.lua`

```
NPL.load("(gl)script/ide/commonlib.lua");

local function StartWebServer()
    local host = "127.0.0.1";
    local port = "8099";
    -- tell NPL runtime to route all HTTP message to the public neuron file `http_
↪server.lua`
    NPL.AddPublicFile("source/SimpleWebServer/http_server.lua", -10);
    NPL.StartNetServer(host, port);
    LOG.std(nil, "system", "WebServer", "NPL Server started on ip:port %s %s", host,␣
↪port);
end
StartWebServer();
```

```
local function activate()
end
NPL.this(activate);
```

filename: `http_server.lua`

```
NPL.load("(gl)script/ide/Json.lua");
NPL.load("(gl)script/ide/LuaXML.lua");

local tostring = tostring;
local type = type;

local npl_http = commonlib.gettable("MyCompany.Samples.npl_http");

-- whether to dump all incoming stream;
npl_http.dump_stream = false;

-- keep statistics
local stats = {
    request_received = 0,
}

local default_msg = "HTTP/1.1 200 OK\r\nContent-Length: 31\r\nContent-Type: text/
↪html\r\n\r\n<html><body>hello</body></html>";

local status_strings = {
    ok ="HTTP/1.1 200 OK\r\n",
    created ="HTTP/1.1 201 Created\r\n",
    accepted ="HTTP/1.1 202 Accepted\r\n",
    no_content = "HTTP/1.1 204 No Content\r\n",
    multiple_choices = "HTTP/1.1 300 Multiple Choices\r\n",
    moved_permanently = "HTTP/1.1 301 Moved Permanently\r\n",
    moved_temporarily = "HTTP/1.1 302 Moved Temporarily\r\n",
    not_modified = "HTTP/1.1 304 Not Modified\r\n",
    bad_request = "HTTP/1.1 400 Bad Request\r\n",
    unauthorized = "HTTP/1.1 401 Unauthorized\r\n",
    forbidden = "HTTP/1.1 403 Forbidden\r\n",
    not_found = "HTTP/1.1 404 Not Found\r\n",
    internal_server_error = "HTTP/1.1 500 Internal Server Error\r\n",
    not_implemented = "HTTP/1.1 501 Not Implemented\r\n",
    bad_gateway = "HTTP/1.1 502 Bad Gateway\r\n",
    service_unavailable = "HTTP/1.1 503 Service Unavailable\r\n",
};
npl_http.status_strings = status_strings;

-- make an HTML response
-- @param return_code: nil if default to "ok"(200)
function npl_http.make_html_response(nid, html, return_code, headers)
    if(type(html) == "table") then
        html = commonlib.Lua2XmlString(html);
    end
    npl_http.make_response(nid, html, return_code, headers);
end

-- make a json response
-- @param return_code: nil if default to "ok"(200)
function npl_http.make_json_response(nid, json, return_code, headers)
    if(type(html) == "table") then
```

```
        json = commonlib.Json.Encode(json)
    end
    npl_http.make_response(nid, json, return_code, headers);
end

-- make a string response
-- @param return_code: nil if default to "ok"(200)
-- @param body: must be string
-- @return true if send.
function npl_http.make_response(nid, body, return_code, headers)
    if(type(body) == "string" and nid) then
        local out = {};
        out[#out+1] = status_strings[return_code or "ok"] or return_code["not_found"];
        if(body~="") then
            out[#out+1] = format("Content-Length: %d\r\n", #body);
        end
        if(headers) then
            local name, value;
            for name, value in pairs(headers) do
                if(name ~= "Content-Length") then
                    out[#out+1] = format("%s: %s\r\n", name, value);
                end
            end
        end
        out[#out+1] = "\r\n";
        out[#out+1] = body;

        -- if file name is "http",  the message body is raw http stream
        return NPL.activate(format("%s:http", nid), table.concat(out));
    end
end


local function activate()
    stats.request_received = stats.request_received + 1;
    local msg=msg;
    local nid = msg.tid or msg.nid;
    if(npl_http.dump_stream) then
        log("HTTP:"); echo(msg);
    end
    npl_http.make_response(nid, format("<html><body>hello world. req: %d. input is %s
↪</body></html>", stats.request_received, commonlib.serialize_compact(msg)));
end
NPL.this(activate)
```

For a full-fledged build-in HTTP server framework in NPL, please see [[WebServer]]

CHAPTER 21

Networking

This topic is FAQ to networking API, please see [[Activation File|ActivationFile]] for real guide of remote communications.

In NPL, you do not need to create any sockets or writing message loops for communication with remote computers. Instead, all communications are managed by NPL very efficiently (see [[NPLArchitecture]] for details). Users only need one single function to establish connection, authenticate and communicate with remote computers.

Initialize Networking Interface

By default, NPL is started without listening on any port, unless you called some library, such as the [[WebServer]]. To enable networking in NPL, one needs to call

```
-- a server that listen on 8080 for all IP addresses
NPL.StartNetServer("0.0.0.0", 8080);
```

The client also needs to call `NPL.StartNetServer`.

```
-- a client that does not listen on any port. Just start the network interface.
NPL.StartNetServer("127.0.0.1", "0");
```

Please see [[Activation File|ActivationFile]] for details.

Server Parameters

The following is from [[WebServer]]'s config file. You can have a general picture of what is configurable in NPL.

```
    <!--HTTP server related-->
    <table name='NPLRuntime'>
      <!--whether to use compression for incoming connections. This must be true in␣
↪order for CompressionLevel and CompressionThreshold to take effect-->
      <bool name='compress_incoming'>true</bool>
      <!---1, 0-9: Set the zlib compression level to use in case compresssion is␣
↪enabled.
      Compression level is an integer in the range of -1 to 9.
          Lower compression levels result in faster execution, but less compression.␣
↪Higher levels result in greater compression,
          but slower execution. The zlib constant -1, provides a good compromise␣
↪between compression and speed and is equivalent to level 6.-->
      <number name='CompressionLevel'>-1</number>
      <!--when the NPL message size is bigger than this number of bytes, we will use␣
↪m_nCompressionLevel for compression.
          For message smaller than the threshold, we will not compress even m_
↪nCompressionLevel is not 0.-->
      <number name='CompressionThreshold'>204800</number>
      <!--if plain text http content is requested, we will compress it with gzip when␣
↪its size is over this number of bytes.-->
      <number name='HTTPCompressionThreshold'>12000</number>
      <!--the default npl queue size for each npl thread. defaults to 500. may set to␣
↪something like 5000 for busy servers-->
      <number name='npl_queue_size'>20000</number>
      <!--whether socket's SO_KEEPALIVE is enabled.-->
      <bool name='TCPKeepAlive'>true</bool>
      <!--enable application level keep alive. we will use a global idle timer to␣
↪detect if a connection has been inactive for IdleTimeoutPeriod-->
      <bool name='KeepAlive'>false</bool>
      <!--Enable idle timeout. This is the application level timeout setting.-->
      <bool name='IdleTimeout'>false</bool>
      <!--how many milliseconds of inactivity to assume this connection should be␣
↪timed out. if 0 it is never timed out.-->
      <number name='IdleTimeoutPeriod'>1200000</number>
      <!--queue size of pending socket acceptor-->
      <number name='MaxPendingConnections'>1000</number>
    </table>
```

Please note, the `IdleTimeout` should be set for both client and server, because timeout may occur on either side, see below for best timeout practice. `TCPKeepAlive` is only for server side.

To programmatically set parameters, see following example:

```
  local att = NPL.GetAttributeObject();
  att:SetField("TCPKeepAlive", true);
  att:SetField("KeepAlive", false);
  att:SetField("IdleTimeout", false);
  att:SetField("IdleTimeoutPeriod", 1200000);
  NPL.SetUseCompression(true, true);
  att:SetField("CompressionLevel", -1);
  att:SetField("CompressionThreshold", 1024*16);
  -- npl message queue size is set to really large
  __rts__:SetMsgQueueSize(5000);
```

TCP Keep Alive

If you want a persistent already-authenticated connection even there are no messages sent for duration longer than 20 seconds, you probably want to generate some kind of regular ping messages. Because otherwise, either the server or the client may consider the TCP connection is dead, and you will lose your authenticated session on the server side.

Although NPL allows you to configure server side application level ping messages globally with `KeepAlive` attribute, it is not recommended to enable it on the server or set its value to a very large value (such as several minutes).

The recommended way is that the client should initiate the ping messages to all remote processes on regular intervals.

For a robust client application, the client also needs to handle connection lost and recover or re-authenticate at application level.

CHAPTER 22

Multi-Threading

In NPL, multi-threading is handled in the same way as networking communication. In other words, activating scripts in other local threads is virtually the same as calling scripts running on another computer. You simply communicate with remote script file with the `NPL.activate` in the same way for both local threads and remote computers. The only difference is that the target script url is different.

For example, to activate a script on local thread `A`, one can use

```
NPL.activate("(A)helloworld.lua", {});
```

To activate the script in a remote computer `B`'s `C` thread, one can use

```
NPL.activate("(C)B:helloworld.lua", {});
```

For more information, please see [[file activation|ActivationFile]]

Creating NPL Worker Thread

NPL worker thread must be created, before messages to it can be processed by script running in that thread.

`NPL.activate("(A)helloworld.lua", {});` does not automatically create thread `A`. You need to call following code to create NPL runtime on thread `A`.

```
NPL.CreateRuntimeState("A", 0):Start();
```

After that, messages sent to `(A)helloworld.lua` will be processed in a real system-level thread called `A`.

Example Project

Now let us create a real multi-threaded application with just a single file `script/test/TestMultithread.lua`. The application print `helloworld` in 5 threads simultaneously.

```
NPL.load("(gl)script/ide/commonlib.lua");

local function Start()
   for i=1, 5 do
      local thead_name = "T"..i;
      NPL.CreateRuntimeState(thead_name, 0):Start();
      NPL.activate(format("(%s)script/test/TestMultithread.lua", thead_name), {
       text = "hello world",
       sleep_time = math.random()*5,
      });
   end
end

local isStarted;
local function activate()
   if(msg and msg.text) then
      -- sleep random seconds to simulate heavy task
      ParaEngine.Sleep(msg.sleep_time);
      LOG.std(nil, "info", "MultiThread", "%s from thread %s", msg.text,  __rts__
↪:GetName());
   elseif(not isStarted) then
      -- initialize on first call
      isStarted = true;
      Start();
   end
end

NPL.this(activate);
```

To run above file, use

```
NPL.activate("(gl)script/test/TestMultithread.lua");
```

or from command line

```
npl script/test/TestMultithread.lua
```

The output will be something like below

```
2016-03-16 6:22:00 PM|T1|info|MultiThread|hello world from thread T1
2016-03-16 6:22:01 PM|T3|info|MultiThread|hello world from thread T3
2016-03-16 6:22:03 PM|T2|info|MultiThread|hello world from thread T2
2016-03-16 6:22:03 PM|T5|info|MultiThread|hello world from thread T5
2016-03-16 6:22:04 PM|T4|info|MultiThread|hello world from thread T4
```

Advanced Examples

Following NPL modules utilize multiple local threads.

- `script/apps/DBServer/DBServer.lua`: a database server, each thread for processing SQL logics, a thread monitor is used to find the most free thread to route any sql query.

Other Options

By design, threading should be avoided to simplify software design and debugging. In addition to real threads, it is usually preferred to use architecture to avoid using thread at all.

- `Timer/callbacks/events/signals` are good candidates for asynchronous tasks in a single thread. With `NPL.activate`, it even allows you to switch implementation without changing any code; and you can boost performance or debug code in a single thread more easily.
- `Coroutine` is a lua language feature, which is also supported by NPL. In short, it uses a single thread to simulate multiple virtual threads, allowing you to share all data in the same thread without using locks, but still allowing the developer to `yield` CPU resource to other virtual threads at any time. The interactive debugging module in NPL is implemented with coroutines. Please see `script/ide/Debugger/IPCDebugger.lua` for details.

CHAPTER 23

Concurrency Model

This is an in-depth topic about NPL's concurrency model and design principles. It also compares it with other similar models in popular languages, like erlang, GO, Scala(java), etc.

What is Concurrency?

The mental picture of all computer languages is to execute in sequential order. Concurrency is a language feature about writing code that runs concurrently. Traditional way of doing this is via threads and locks, which is very troublesome to write. The Actor Model, which was first proposed by Carl Hewitt in 1973, takes a different approach to concurrency, which should avoid the problems caused by threading and locking.

There are many implementations of concurrency model in different languages, they differ both in performance under different use cases, and in the programmers' mental picture when writing concurrent code.

> NPL uses a hybrid approach, which give you the ability to run tens of thousands of tasks in a single thread or across multiple threads. More importantly, you do not need to write any code to spawn a virtual process or write error-prone message loops. In short, NPL is very fast, scalable and give programmers a mental picture that is close to neurons in the brain.

Concurrent Activation in NPL

Preemptive vs Non-Preemptive File Activation

By default NPL activate function is non-preemptive, it is the programmer's job to finish execution within reasonable time slice. The default non-preemptive mode gives the programmer full control of how code is executed and different neuron files can easily share data using global tables in the same thread.

On the other hand, NPL also allows you to do preemptive activation, in which the NPL runtime will count virtual machine instructions until it reaches a user-defined value (such as 1000) and then automatically yield(pause) the activate function. The function will be resumed automatically in the next time slice. NPL time slice is by default about 16ms (i.e. 60FPS).

To make file activate function preemptive, simply pass a second parameter `{PreemptiveCount, MsgQueueSize, [filename|name], clear}` to `NPL.this` like below:

- PreemptiveCount: is the number of VM instructions to count before it yields. If nil or 0, it is non-preemptive. Please note JIT-compiled code does not count as instruction by default, see below.
- MsgQueueSize: Max message queue size of this file, if not specified, it is same as the NPL thread's message queue size.
- filename|name: virtual filename, if not specified, the current file being loaded is used.
- clear: clear all memory used by the file, including its message queue. Normally one never needs to clear. A neuron file without messages takes less than 100 bytes of memory (mostly depending on the length's of its filename)

```
-- this is a demo of how to use preemptive activate function.
NPL.this(function()
    local msg = msg; -- important to keep a copy on stack since we go preemptive.
    local i=0;
    while(true) do
        i = i + 1;
        echo(tostring(msg.name)..i);
        if(i==400) then
            error("test runtime error");
        end
    end
end, {PreemptiveCount = 100, MsgQueueSize=10, filename="yourfilename.lua"});
```

You can test your code with:

```
NPL.activate("yourfilename.lua", {name="hi"});
```

Facts about preemptive activate function:

- It allows you to run tens of thousands of jobs concurrently in the same system thread. Each running job has its own stack and the memory overhead is about 450bytes. A neuron file without pending messages takes less than 100 bytes of memory (mostly depending on the length's of its filename). The only limitation to the number of concurrent jobs is your system memory.
- There is a slight performance penalty on program speed due to counting VM instructions.
- With preemptive activate function, the programmer should pay attention when making changes to shared data in the thread, since your function may be paused at any instruction. The golden rule here is never make any changes to shared data, but use messages to exchange data.
- C/C++ API call is counted as one instruction, so if you call ParaEngine.Sleep(10), it will block all concurrent jobs on that NPL thread for 10 seconds.
- Code in async callbacks (such as timer, remote api call) in activate function are NOT preemptive. Because callbacks are invoked from the context of other concurrent activate functions.

Test Cases and Examples:

see also `script/ide/System/test/test_concurrency.lua` for more tests cases.

```
local test_concurrency = commonlib.gettable("System.Core.Test.test_concurrency");

function test_concurrency:testRuntimeError()
    NPL.this(function()
        local msg = msg; -- important to keep a copy on stack since we go preemptive.
        local i=0;
        while(true) do
            i = i + 1;
            echo(tostring(msg.name)..i);
            if(i==40) then
                error("test runtime error");
            end
        end
    end, {PreemptiveCount = 100, MsgQueueSize=10, filename="tests/testRuntimeError"});
    NPL.activate("tests/testRuntimeError", {name="1"});
    NPL.activate("tests/testRuntimeError", {name="1000"});
end

function test_concurrency:testLongTask()
    NPL.this(function()
        local msg = msg; -- important to keep a copy on stack since we go preemptive.
        local i=0;
        while(true) do
            i = i + 1;
            echo(i);
        end
    end, {PreemptiveCount = 100, MsgQueueSize=10, filename="tests/testLongTask"});
    NPL.activate("tests/testLongTask", {name="1"});
end

function test_concurrency:testMessageQueueFull()
    NPL.this(function()
        local msg = msg; -- important to keep a copy on stack since we go preemptive.
        local i=0;
        for i=1, 1000 do
            i = i + 1;
        end
        echo({"finished", msg});
    end, {PreemptiveCount = 100, MsgQueueSize=3, filename="tests/testMessageQueueFull
↪"});
    for i=1, 10 do
        NPL.activate("tests/testMessageQueueFull", {index=i});
    end
    -- result: only the first three calls will finish.
end

function test_concurrency:testMemorySize()
    __rts__:SetMsgQueueSize(100000);
    for i=1, 10000 do
        NPL.this(function()
            local msg = msg; -- important to keep a copy on stack since we go␣
↪preemptive.
            for i=1, math.random(1,1000) do
                msg.i = i;
            end
            echo(msg);
        end, {PreemptiveCount = 10, MsgQueueSize=1000, filename="tests/testMemorySize
↪"..i});
        NPL.activate("tests/testMemorySize"..i, {index=i});
    end
    -- TODO: use a timer to check when it will finish.
end
```

```
function test_concurrency:testThroughput()
    __rts__:SetMsgQueueSize(100000);
    for i=1, 10000 do
        NPL.this(function()
            local msg = msg;
            while(true) do
                echo(msg)
            end
        end, {PreemptiveCount = 10, MsgQueueSize=3, filename="tests/testThroughput"..
↪i});
        NPL.activate("tests/testThroughput"..i, {index=i});
    end
end
```

NPL Message Scheduling

Each NPL runtime can have one or more NPL states/threads (i.e. real system threads). Each NPL state has a single message queue for input and output with other NPL threads or remote processes. Programmer can set the maximum size of this queue, so when it is full, messages are automatically dropped.

On each time slice, NPL state will process `ALL` messages in its message queue in priority order.

- If a message belongs to a non-preemptive file, it will invoke its activate function immediately.
- If a message belongs to a preemptive file, it will remove the message from the queue and insert it to the target file's message queue, which may has a different message queue size. If message queue of the file is full, it will drop the message immediately.

In another native thread timer, all active preemptive files (with pending/half-processed messages) will be processed/resumed in `preemptive` way. I.e. we will count VM(virtual machine) instructions for each activate function and pause it if necessary.

> Note: all TCP/IP network connections are managed by a single global network IO thread. In this way you can have tens of thousands of live TCP connections without using much system memory. A global NPL dispatcher automatically dispatch incoming network messages to the message queue of the target NPL state/thread. Normally, the number of NPL threads used is close to the number of CPU cores on the computer.

Causions on Preemptive Programming

Since preemptive code and non-preemptive code can coexit in the same NPL state (thread). It is the programmers' job to ensure preemptive code does not modify global data.

- Note: when NPL debugger is attached, all preemptive code are paused to make debugging possible.
- Also, please note that JIT-compiled code does NOT call hooks by default. Either disable the JIT compiler or edit src/Makefile: `XCFLAGS= -DLUAJIT_ENABLE_CHECKHOOK` This comes with a speed penalty even hook is not set. Our recommendation is to call `dummy()` or any NPL API functions and count that.

Priority of Concurrent Activation

PreemptiveCount is the total number of instructions (or Byte code) executed in an activate function before we pause it. In NPL, `PreemptiveCount` can be specified per activate function, thus giving you fine control over how much computation each activate function can run in a given time slice.

The NPL scheduler for preemptive activation file will guarantee each function will run precisely PreemptiveCount instructions after PreemptiveCount * total_active_process_count instructions every time slice.

So there is absolutely no dead-lock conditions in our user-mode `preemptive` scheduler. The only thing that may break your application is running out of memory. However, each active (running) function only take 400-2000 bytes according to usage. even 1 million concurrently running jobs takes only about 1GB memory. If only half of those jobs are busy doing calculation on average, you get only a little over 500MB memory usage.

Language Compare: The Mental Picture

We will compare concurrency model in several languages in terms of programmer's mental picture and implementation.

Erlang

In erlang, one has to manually call `spawn` function to create a user-mode virtual process. So it is both fast and memory efficient to create millions of erlang processes in a single thread. Erlang simulates `preemptive` scheduling among those processes by counting byte code executed. Erlang processes are currently scheduled on a reduction count basis as described here and here. One reduction is roughly equivalent to a function call. A process is allowed to run until it pauses to wait for input (a message from some other process) or until it has executed 1000 reductions. A process waiting for a message will be re-scheduled as soon as there is something new in the message queue, or as soon as the receive timer (receive ... after Time -> ... end) expires. It will then be put last in the appropriate queue. Erlang has 4 scheduler queues (priorities): 'max', 'high', 'normal', and 'low'.

- Pros: In the viewpoint of programmers, erlang process is `preemptive`. Underneath it is not true `preemptive`, but as along as it does not call `C` functions, the scheduler is pretty accurate.
- Cons: The programmers need to manually name, create and manage the process's message loop (such as when message queue grows too big).

Comparison:

- Similarity:
  - Both NPL and erlang copies messages when sending them
  - Both support `preemptive` concurrent scheduler.
  - Both use user mode code to simulate processes, so that there is almost no limit to the total number of asynchronous entities created.
- Differences:
  - Erlang is `preemptive`; NPL can be not `preemptive` and `non-preemptive`.
  - NPL use source code file name as the entity name, and it does not require the programmer to create and manually write the message loop. Thus the mental picture is `each NPL file has a hidden message loop`.
  - Erlang does not give you explicit control over which real system thread your process is running (instead, it automatically does it for you). In NPL, each neuron file can be explicitly instanced on one or more system-level thread.
  - Erlang scheduler does not give you control over Instruction Count per process before it is preemptied. In NPL, one can specify different instruction count per activate function, thus giving you a much finer control over priority.

  - In NPL we count each C API as a single instruction, while Erlang tries to convert C API to the number of ByteCode executed in the same time.

Go

To me, GO's concurrency model is a wrapper of C/C++ threading model. Its syntax and library can greatly simply the code (if written in C/C++). It uses real system threads, so you can not have many concurrent jobs like erlang or NPL.

Comparison:

- NPL can be written in the same fashion of GO, because it give you explicit control over real system-level thread.

Scala

Scala is a meta language over higher general purpose language like java. Scala is like a rewrite of Erlang in java. Its syntax is in object-oriented fashion with a mixture of Erlang's functional style. So please refer to erlang for comparision. However, Scala is not-preemptive as erlang, it relies on the programmer to yield in its event based library, while NPL supports `preemptive` mode in additional to `non-preemptive` mode.

Final Words

- Unlike erlang, go, scala, NPL is a dynamic and weak type language. In NPL, it is faster to invoke C/C++ code, its byte code is as fast as most strongly typed languages. See [[NPLPerformance]] for details.

CHAPTER 24

NPL Common Libraries

NPL common Libraries contains a rich and open source collection of libraries covering: io, networking, 2d/3d graphics, web server, data structure, and many other software frameworks.

Library Installation

NPL common libraries can be installed in `pkg` or `zip` or in plain source code. When you deploy your application, you can choose to use our pre-made zip file or deploy only used files in a zip package by yourself.

If you installed NPL runtime via ParaCraftSDK on windows platform, you will already have NPL common library installed in `ParaCraftSDKGit/NPLRuntime/win/packages/main.pkg`.

If you installed NPL runtime from source code on linux. It does not have NPL common libraries preinstalled. Instead you need to copy or link the `script` folder to your project's working directory. The folder to link to can be found here, which contains thousands of open source and documented files all written in NPL.

Major Libraries

For minimum server-side application development, include this:

```
NPL.load("(gl)script/ide/commonlib.lua");
```

or

```
NPL.load("(gl)script/ide/System/System.lua");
```

For full-fledged heavy 2d/3d application development, include this:

```
NPL.load("(gl)script/ide/IDE.lua");
```

or

```
NPL.load("(gl)script/kids/ParaWorldCore.lua");
```

Documentation

NPL libraries are themselves written in pure NPL scripts. They usually have no external dependencies, but on the low level NPL API provided by NPL runtime. These API is in turn implemented in C/C++ in cross-platform ways.

The documentation for low level API is here [[https://codedocs.xyz/LiXizhi/NPLRuntime/modules.html]]

However, it is not particularly useful to read low level API. All you need to do it is to glance over it, and then dive into the source code of NPL libraries here.

CHAPTER 25

Object Oriented Programming

All NPL library source code is written in object oriented way. It is recommended that you do the same for your own code.

> Lua itself can be used as a functional language, which does not enforce object oriented programming. It is the programmer's choice of how to write their code.

NPL has a number of library files to help you code in object oriented ways.

Referencing a class

`commonlib.gettable` and `NPL.load` is the primary way for you to include and import other component. Because all NPL libraries are defined to be compatible with `commonlib.gettable`, it is possible to import the class namespace table without loading it. This also makes the order of importing or loading libraries trivial.

Remember that NPL/Lua is a statically scoped language and each function is hash value on its containing table, it is important to cache class table on a local variable.

Example of importing `commonlib.LinkedList` into `LinkedList`.

```lua
local LinkedList = commonlib.gettable("commonlib.LinkedList");

local MyClass = commonlib.gettable("MyApp.MyClass");
function MyClass.Test()
   local list = LinkedList:new();
end
```

By the time, you call the function on the imported table, you need to `NPL.load` its implementation before hand. As you see, most common class is already buddled in common include files, such as `NPL.load("(gl)script/ide/commonlib.lua");`, this include the implementation of `commonlib.gettable` as well. If you are interested, you can read its source code. So if you have loaded that file before, such as in your `bootstrapper` file, you do not need to load it in every other files. But you need to call `commonlib.gettable` on the beginning of every file using the given library, for better performance.

It is good practice to only NPL.load non-frequently used files shortly before they are used. So that, your application can boot up faster, and use less memory.

Defining a singleton class

For singleton class, which contains only static functions. It is sufficient to use commonlib.gettable to define the class. The idea is that you reference the class,and then add some static implementations to it dynamically.

```
local MyClass = commonlib.gettable("MyApp.MyClass");

function MyClass.Method1()

end

function MyClass.Method2()

end
```

Defining an ordinary class

For class, which you want to create instances from, you need to use the commonlib.inherit function. For implementation of commonlib.inherit, please see script/ide/oo.lua.

The following will define a MyClass class with new method.

```
local MyClass = commonlib.inherit(nil, commonlib.gettable("MyApp.MyClass"));

MyClass.default_param = 1;

-- this is the constructor function.
function MyClass:ctor()
   self.map = {};
end

function MyClass:init(param1)
   self.map.param1 = param1;
   return self;
end

function MyClass:Clone()
   return MyClass:new():init(self:GetParam());
end

function MyClass:GetParam()
   return self.map.param1;
end
```

To create a new instance of it

```
local MyClass = commonlib.gettable("MyApp.MyClass");

local c1 = MyClass:new():init("param1");
local c2 = MyClass:new():init("param1");
```

You can define a derived class like below

```
local MyClassDerived = commonlib.inherit(commonlib.gettable("MyApp.MyClass"),␣
↪commonlib.gettable("MyApp.MyClassDerived"));

-- this is the constructor function.
function MyClassDerived:ctor()
   -- parent class's ctor() have been automatically called
end

function MyClassDerived:init(param1)
   MyClassDerived._super.init(self, param1);
   return self;
end
```

Advanced `ToolBase` class

If you want a powerful class with `event/signal/auto property`, and `dynamic reflections`, you can derive your class from `ToolBase` class.

See the examples.

```
local Rect = commonlib.gettable("mathlib.Rect");

-- class new class
local UIElement = commonlib.inherit(commonlib.gettable("System.Core.ToolBase"),␣
↪commonlib.gettable("System.Windows.UIElement"));
UIElement:Property("Name", "UIElement");
UIElement:Signal("SizeChanged");
UIElement:Property({"enabled", true, "isEnabled", auto=true});


function UIElement:ctor()
    -- client rect
    self.crect = Rect:new():init(0,0,0,0);
end

function UIElement:init(parent)
    self:SetParent(parent);
    return self;
end

-- many other functions omitted here
```

CHAPTER 26

Deploy Your Application

You can deploy your application to windows, linux, ios, android, etc. Both 32bits/64bits versions are supported.

- NPL scripts can be deployed in plain `*.lua or *.npl` text files or in precompiled `*.o` binary file.
- You can also buddle all or part of your scripts or any other read-only resource files into one or more zipped archive files.
- You can deploy NPL runtime globally or side-by-side with your application files.

For automatic deployment, please install and use `ParacraftSDK`. This articles explains how to do it manually.

Pre-compiling NPL script

Precompiled NPL script is called bytecode in lua. The bytecode generated by `Luajit` is incompatible with bytecode generated by `lua`, but is cross-platform for any (32bits/64bits) architecture. If you choose to deploy your app with bytecode, you must also choose to use luajit or lua when deploying NPL runtime.

Please read the documentation in `NPLCompiler.lua` for more information. Examples:

```
NPL.load("(gl)script/ide/Debugger/NPLCompiler.lua");
NPL.CompileFiles("script/*.lua", nil, 100);
```

Buddle Scripts in Zip files

You can buddle your script and assets in zip files. There are two kinds of zip files: one is the standard `*.zip` file, the other is called `*.pkg` file. You can create `*.pkg` file from a standard zip file with the NPL runtime like below. `*.pkg` use a simple encription algorithm over the zip file.

```
ParaAsset.GeneratePkgFile("main.zip", "main.pkg");
```

When application starts, NPL runtime will automatically load all `main*.pkg` and `main*.zip` files in the application's start directory into memory. The load order is based on file name, so that the a file in "main_patch2.pkg" will overwrite the same file in "main_patch1.pkg".

Please note that loading `pkg` file is very fast, it only copys the file into memory, individual script file or assets are only unzipped and parsed on first use.

A programmer can also programmatically load or unload any archive file using the NPL API like below.

```
NPL.load("pluginABC.pkg");
-- or using explicit calls
ParaAsset.OpenArchive("pluginABC.pkg", true);
```

The second parameter is whether to use relative path in archive files. (i.e. file path in archive file are relative to the containing directory). Search paths, such as from [[npl_packages]] are honored when loading archives.

Deploy NPLRuntime Side-By-Side

Deploying NPL Runtime side-by-side is as easy as copying all executable files to the application directory. The recommended deployment folder structures is below

```
bin/: npl exe, dll, lua,luajit, etc
packages/: common *.pkg *.zip package files
script/: your own script files
config.txt
any other files
```

Another way is to deploy everything to the application root directory.

```
script/: your own script files
npl exe, dll, lua,luajit, etc
common *.pkg *.zip package files
config.txt
any other files
```

> if `config.txt` file is on the root application directory. Its cmdline content will be appended to NPL command line when running NPL runtime from this directory.

An example `config.txt`, see below:

```
cmdline=noupdate="true" debug="main" bootstrapper="script/apps/HelloWorld/main.lua"
```

Luajit vs Lua

Luajit and Lua are ABI-compatible, meaning that you can deploy NPL runtime with them simply by replacing `lua.dll(so)` with either implementation. Luajit is the recommended way for release deployment on all platforms. However, currently on iOS, Luajit is used but disabled, since it is not allowed. Even a disabled luajit runs faster than lua and its bytecode is the same for both 32bits and 64bits OS. So you would only want to deploy lua dll for debugging purposes on development machines.

CHAPTER 27

NPL Packages

NPL Package is a special folder under `npl_packages/[package_name]/`. Files in it are always organized as if they are relative to the working directory. So this folder can be used as a search path directly, or zipped to a `*.zip|pkg` to be used as an archive file, or loaded by file module name all at the same time.

NPL package serves following purposes:

- it provides a way to release your software modules to be used by someone else.
- it can be used as additional search path for third-party plugins, which I will explain later.
- it provides a way to allow different versions of the same file module to coexist by putting them in different npl packages in a single application. See [[LoadFile]]
- it provides a way to install third-party modules at development time.

Package As Search Path

Files in `npl_packages/[package_name]` are usually relative to the root working directory.

Therefore, developer who wants to use other people's modules can simply add `npl_packages/[package_name]` to global search path by calling following code:

```
NPL.load("npl_packages/some_test_module/")
```

The trick is to end with `/` in file name. What this function does is actually find the package folder in a number of possible locations (see next section), and then add it to the global search path. By default, when NPL starts, it will always try to load the official 'npl_packages/main/' package. So you do not need to call `NPL.load("npl_packages/main/")` in order to use the rich set of open source NPL libraries.

Another way is to load via command line, such as below. See [[NPLCommandLine]]

```
npl loadpackage="npl_packages/paracraft/" dev="/"
```

How NPL Locate Packages

NPL locates package folder given by its relative path in following order:

- search in current working directory
- search in current executable directory
- search recursively for 5 parent directories of the executable directory.

For example, suppose:

- your current working directory is `/home/myapp/`,
- and your executable directory is `/opt/NPLRuntime/redist/bin64/`,

then `NPL.load("npl_packages/main/")` will search following directories until one exists and add it to the search path.

- /home/myapp/npl_packages/main/
- /opt/NPLRuntime/redist/bin64/npl_packages/main/
- /opt/NPLRuntime/redist/npl_packages/main/
- /opt/NPLRuntime/npl_packages/main/
- /opt/npl_packages/main/
- /npl_packages/main/

What Happened After Loading A Package

The short answer is `nothing happens`, because loading a package only add its folder to the global search path. You still need to load any module files in the package folder with `NPL.load`. For example, support you have `NPL.load("npl_packages/test/")` successfully loaded at '/home/myapp/npl_packages/test/' and there is module file at `/home/myapp/npl_packages/test/script/any_module_folder/test.lua` Then you can load it with relative file path. `NPL.load("script/any_module_folder/test.lua")` However, if there is a file at your working directory such as `/home/myapp/script/any_module_folder/test.lua`, this file will be loaded rather than the one in the global search path.

File Search Order

Files in current working directory are always searched first (including those in zipped archive files) before we resolve to global search path. Moreover, search path added last is actually searched first. There is one exception, if NPL is run with dev directory in its command line `dev="dev_folder"`, NPL packages in `dev_folder` are loaded before zipped archive files. This allows one to use the latest source code during development.

For example:

```
NPL.load("npl_packages/A/")
NPL.load("npl_packages/B/")
NPL.load("test.lua");
```

`test.lua` is searched first in current working directory and any loaded archive (zip, pkg) file. If not exist, it will search in `npl_packages/B/` and then in `npl_packages/A/`.

Usage

For package developers:

NPL packages are mostly used at development time by other developers (not end users).

i.e. If you want other developers' to use your code, you can upload your working directory to git, so that other developers can clone your project to their `npl_packages/[your module name]`. Package can include not only source code, but also binary asset files, like images, textures, sound, 3d models, etc.

For other developers:

Other developers should merge all used npl_packages to working directory, before releasing their software.

i.e. It is the developer's job to resolve dependencies, in case multiple versions of the same file exist among used npl_packages. At release time, it is recommended NOT to redistribute the npl_package folder, but copy/merge the content in them to the working directory, pre-compile all source code and package code and/or assets in one or multiple archive files. Please see the [[DeployGuide]] for details. However, if different versions of the same file must coexist, we can use file-based modules to distribute in separate `npl_packages` folders.

Where To Find NPL Packages

Each repository under NPLPackages is a valid npl_package managed by the community.

Click here for more details on NPL packages.

If one want to upload their own package here, please make an issue here, and provide links to its code.

How To Install A Package

Simply create a folder under your development's working directory, create a sub folder called `npl_packages`. And then run git clone from there. Like this:

```
cd npl_packages
git clone https://github.com/NPLPackages/main.git
```

See also paracraft package for another example.

File-based Modules

See [[LoadFile]].

How to Contribute

It is NOT advised to modify or add files in the ./npl_packages folder, instead create a similar directory structure in your project's development directory if you want to add or modify package source code. If you do want to contribute to any npl packages, please fork it on github and send pull requests to its author on github.

For example, if you want to modify or add a file like `./npl_packages/main/.../ABC.lua` Instead of modify it in the npl package folder, you simply create a file at the root development folder with the same directory structure like this `./.../ABC.lua`. At runtime, your version of file will be loaded instead of the one in npl package folder.

When your code is mature, you may consider fork the given npl_package in another place, and merge your changed files and send the author a pull request. If the author responds fast, he or she may accept your changes and you can later get rid of your changed files in your original project.

CHAPTER 28

Meta Programming in NPL

Meta programming allows you to extend NPL syntax using the NPL language itself. For example, following code are valid in NPL.

```
local a=1;
loop(){  please execute the code 10 times with i
    echo(a+i)
    async(){
        echo("This is from worker thread");
    }
}
```

Here loop, async are extended NPL syntax defined elsewhere.

The concept is first developed by LISP programming language in 1960s. However, despite the power of LISP, its syntax is hard to read for most programmers nowadays. NPL introduces a similar concept called `Function-Expression`, which can be mixed very well with the original NPL syntax.

The syntax of NPL's Function-Expression is `name(input, ...){ ... }`

Files with `*.npl` extension support Function-Expression syntax by default. For example

```
NPL.load("(gl)script/tests/helloworld.npl")
NPL.loadstring("-- source code here", "filename_here")
```

Function-Expression User Guide

`def` expression

```
def(<name>, <params>){
    --mode:<mode>
    statements
}
```

- <name>:name of new structure to be defined, name could not be null
- <params>:parameters to be passed to defined structure multiple parameters should be seperated by comma unknown number parameters, using ...
- <mode>:mode is set at the first line of comment inside block. Mode could be strict, line and token. When different mode is set, different parsing strategy is used in defined function expression. If nothing specified, strict mode is used.
- <statements>: statements here are template code, which would be applied to final code without any change. However, one exception is +{} structure. Code inside +{} would be executed during compiling. And some default functions are provided to give users more control.

Default functions in +{}*emit(str, l)*: emit str at the line l. when no str is specified, it emit whole code chunk inside function expression block. when no l is specifiec, it emit at first line*emitline(fl, ll)*: used only in line mode. emit code chunk from line fl to ll. If no ll specified it emit from fl to end of code chunk*params(p)*: emit parameter p

UsageAfter defining a function expression, it could be used like this:

```
<name>(<params>){
     statements in specified mode
}
```

Mode

- strict: statements in strict mode are followed the rules of original npl/lua grammar
- line: statements in line mode are treated as lines, no grammar and syntax rules
- token: statements in token mode are treated as token lists, original symbols and keywords are kept, but no grammar and syntax rules

Examples

Example 1

```
def("translate", x, y, z){
    push()
    translate(+{params(x)}, +{params(y)}, +{params(z)})
    +{emit()}
    pop()
}

translate(1,2,3){
    rotate(45)
}
```

The above chunk will compiled into

```
push()
translate(1,2,3)
rotate(45)
pop()
```

Example 2

```
def("loop"){
  --mode:line
  +{local line = ast:getLines(1,1)
    local times, i = line:match("execute the code (%w+) times with (%l)")
    if not times then times="1" end
    if not i then i="i" end
       }
 for +{emit(i)}=1, +{emit(times)} do
 +{emitline(2)}
 end
 }

 loop(){execute the code 10 times with j
    print(2+j)
    print(3)
    print(4)
 }
```

The above chunk will compiled into

```
do for j=1, 10 do
print(2+j)
print(3)
print(4) end end
```

For more examples, please see our test here or dsl definition file here

CHAPTER 29

NPL C/C++ Architecture

This section covers cross-platform modules written in C/C++. These modules are exposed via NPL scripting API so that they are called via NPL.

C++ Modules

NPL Scripting Engine:

- NPL state (or NPL virtual code environment): a single NPL thread, has its own memory allocators and manages all files it load.
  - NPL state can load NPL/Lua script or C++ dll.
  - Mono state: can load C# dll.
- NPL Networking: Manage all local or remote NPL states via NPL HTTP/TCP connections.

Graphics Engine:

All GUI objects must be created in the main NPL thread, which is the same as renderer thread. 2D/3D Engine are all object oriented. All 2D objects are origanized in a parent/child tree. 3D objects are organized in a quad-tree in additional to parent/child tree.

Video/audio Renderer

A static/dynamic buffer, textures, draw 3d/2d api, fonts, etc.

- DirectX fully supported renderer.
- OpenGL renderer: a cross-platform renderer(limited functions used in our linux/android build)

PaintEngine

It is like a GDI engine used by 2D Engine for drawing 2D and simple 3d objects, like lines, rectangles, etc.

2D Engine: GUIBase is the base class to all 2d object.

- GUIRoot: root node of all GUI objects.
- GUIContainer, GUIButton, GUIText, etc

3D Engine: BaseObject is the base class to all 3D object.

- ViewportManager: manages 2D viewport into which we can render 3d or 2d objects.
- SceneObject: root node of all 3D objects. It manages all objects like adding, deleting, searching, rendering, physics, etc
- TerrainTileRoot: it is a `quadtree` container of all 3D objects for fast object searching according to their 3d locations and current camera frustum.
- CameraObject: camera frustum. several derived class like AutoCamera, etc
- MeshObject: base class static triangle mesh objects in the 3d object.
- MeshPhysicsObject: it is a static mesh with physics.
- BipedObject: it represents an animated object.
- MinisceneGraph: it is a simplified version of SceneObject, which usually manages a small number of 3d objects which can be rendered separately in to a 2D texture.
- TerrainEngine: infinitely large terrain with a heightmap, and multiple texture layers. It is rendered with dynamic level-of-detail algorithm.
- Other Scene objects:
- [[BlockEngine]]: it manages rendering of `32000x32000x256` blocks.
  - BlockRegion: manages `512x512x256` blocks, which are saved into a single file
  - ChunkColumn: `16x16x256`
  - Chunk: `16x16x16`, a static renderable object.

Physics Engine

It uses the open source Bullet physics engine for static mesh objects in the scene. Block physics is handled separately by the [[BlockEngine]] itself.

Asset Management:

Usually each static asset file is a asset entity. `AssetEntity` is the based class to all assets in the system. Assets provides data and sometimes rendering methods to be used any other 2d/3d objects. All assets are loaded asynchronously by default from either IO disk or remote network.

- TextureEntity: 2d textures
- MeshEntity: static mesh file like `.x`

- ParaXEntity: animated mesh file which can be loaded from `.x`, `.fbx`
- Audio, bmax model, database, font, etc.
- EffectFile: shader files, which can be loaded from directX `.fx`.

IO and Util

- FileManager: manages all files and search paths.
- CParaFile: manages a single file read/write.
- math: 2d/3d math libs like vectors and matrices

ParaScripting API:

- C++ –> NPL: exposing all above functions and modules to NPL scripting environment.
- See all C++ bindings in NPL scripting reference

CHAPTER 30

Core ParaEngine/NPL API

The following API is implemented in C/C++ and exposed to NPL.

- Core API Module Reference

  For examples of using these NPL API, please refer to source code of NPL packages, like the main package

In NPL scripts, tables that begins with `Para` like `ParaIO`, `ParaUI`, `ParaScene`, `ParaEngine` are usually core C++ API. Moreover, most functions in `NPL` table is also core C++ API, like `NPL.load`, `NPL.this`, etc.

CHAPTER 31

Attribute System

Almost all C++ API objects like `ParaUIObject`, `ParaObject`, and even some global table like `ParaEngine`, expose a data interface via `ParaAttributeObject`.

The attribute system allows us to easily get or set data in core C++ object via NPL scripts, like below.

```
local attr = ParaEngine.GetAttributeObject();
local value = attr:GetField("IgnoreWindowSizeChange", false);
attr:SetField("IgnoreWindowSizeChange", not value);
```

In NPL code wiki, one can open from menu `view::object browser` to inspect all living core objects via the attribute system.

> Please note script/ide/System/Core/DOM.lua provides a handy interface to iterate over existing core objects or even pure NPL tables.

CHAPTER 32

Asset Manifest & Asynchronous Asset Loading

A graphical application usually depends on tons of assets (textures, models, etc) to process and render. It is usually not possible deploy all assets to client machine at installation time, instead assets are downloaded from the server usually on first use.

NPL/ParaEngine has built-in support for asynchronous asset loading via asset manifest system. Basically it resolves two problems:

1. Use a plain text file to look up and download assets in the background in several worker threads.
2. Most UI and 3d objects in ParaEngine provide a synchronous interface to set assets, as if they are already available. In reality, they will automatically resolves assets (also their dependencies) when those assets are available locally.

Asset Manifest Manager

When an application starts, NPLRuntime will read all `Assets_manifest*.txt` file under the root directory. Each file has following content

```
format is [relative path],md5,fileSize
```

if the name ends with .z, it is zipped. This could be 4MB uncompressed in size md5 is checksum code of the file. fileSize is the compressed file size.

```
audio/music.mp3.z,3799134715,22032
model/building/tree.dds.z,2957514200,949
model/building/tree.x.z,2551621901,816
games/tutorial.swf,1157008036,171105
```

When one of the async loader try to load an application asset(texture, model, etc), it will first search in AssetManifest using the TO-LOWER-CASED asset path, such as (model/building/tree.x). if will then search the "temp/cache/" directory for a matching file

The file matching is done by comparing the line in the asset file with the filename in the cache directory, using their md5 and size.

```
audio/music.mp3.z,3799134715,22032 matches to file 379913471522032
```

Example Usage:

```
AssetFileEntry* pEntry = CAssetManifest::GetSingleton().GetFile("Texture/somefile.
↪dds");
if(pEntry && pEntry->DoesFileExist())
{
    // Load from file pEntry->GetLocalFileName();
}
```

CHAPTER 33

ParaObject

ParaObject is the scripting proxy to a 3D object on the C++ engine. In most cases, it could be a 3d mesh, an animated character called biped, a bmax model or any custom 3D objects.

Create 3D Object

Use `CreateCharacter` to create animated character based on a ParaX asset file.

```
local player = ParaScene.CreateCharacter ("MyPlayer", ParaAsset.LoadParaX("",
↪"character/v3/Elf/Female/ElfFemale.x"), "", true, 0.35, 0, 1.0);
player:SetPosition(ParaScene.GetPlayer():GetPosition());
ParaScene.Attach(player);
```

Use `CreateMeshPhysicsObject` to create a static mesh object based on a mesh asset file

```
local asset = ParaAsset.LoadStaticMesh("","model/common/editor/z.x")
local obj = ParaScene.CreateMeshPhysicsObject("blueprint_center", asset, 1,1,1, false,
↪ "1,0,0,0,1,0,0,0,1,0,0,0");
obj:SetPosition(ParaScene.GetPlayer():GetPosition());
obj:GetAttributeObject():SetField("progress",1);
ParaScene.Attach(obj);
```

Get View Parameters

Please note all asset are async-loaded, when asset is not loaded, object renders nothing, and following parameters may not be correct when associated mesh asset is not async-loaded.

```
local obj = ParaScene.GetObject("MyPlayer")
local params = {};
param.rotation = obj:GetRotation({})
param.scaling = obj:GetScale();
param.facing = obj:GetFacing();
param.ViewBox = obj:GetViewBox({});
local x,y,z = obj:GetViewCenter();
```

References:

More Complete API reference, please see ParaObject Reference

CHAPTER 34

System Library

All system library is contained in the main package. See [[here|npl_packages]] for how to install packages.

Commonly used packages

- Timer: showdoc
- Serialization
- Database
- HTTP
- Networking
- ... TODO

CHAPTER 35

Timer

- **Encoding:** script/ide/timer.lua

```
NPL.load("(gl)script/ide/timer.lua");

local mytimer = commonlib.Timer:new({callbackFunc = function(timer)
    commonlib.log({"ontimer", timer.id, timer.delta, timer.lastTick})
end})

-- start the timer after 0 milliseconds, and signal every 1000 millisecond
mytimer:Change(0, 1000)

-- start the timer after 1000 milliseconds, and stop it immediately.
mytimer:Change(1000, nil)

-- now kill the timer.
mytimer:Change()

-- kill all timers in the pool
commonlib.TimerManager.Clear()

-- dump timer info
commonlib.TimerManager.DumpTimerCount()

-- get the current time in millisecond. This may be faster than ParaGlobal_
↪timeGetTime() since it is updated only at rendering frame rate.
commonlib.TimerManager.GetCurrentTime();

-- one time timer
commonlib.TimerManager.SetTimeout(function()  end, 1000)
```

CHAPTER 36

Serialization, Encoding and Logging

- Serialization: script/ide/serialization.lua
- Encoding: script/ide/Encoding.lua
- Logging: script/ide/log.lua
- SHA1: script/ide/System/Encoding/sha1.lua

NPL/Lua Table Serialization

see `script/test/TestNPL.lua`

```
local o = {a=1, b="string"};

-- serialize to string
local str = commonlib.serialize_compact(o)
-- write string to log.txt
log(str);
-- string to NPL table again
local o = NPL.LoadTableFromString(str);
-- echo to output any object to log.txt
echo(o);
```

Data Formatting

```
log(string.format("succeed: test case %.3f for %s\r\n", 1/3, "hello"));
-- format is faster than string.format, but only support limited place holder like %s␣
↪and %d.
log(format("succeed: test case %d for %s\r\n", 1, "hello"));
```

Data Encoding

- Encoding: script/ide/Encoding.lua

```
commonlib.echo(NPL.EncodeURLQuery("http://www.paraengine.com", {"name1", "value1",
↪"name2", "",}))
```

UTF8 vs Default Text Encoding

Encoding is a complex topic. The rule of thumb in NPL is to use utf8 encoding wherever possible, such as in source code, XML/html/page files, UI text, network messages, etc. However, system file path must be encoded in the operating system's default encoding, which may be different from utf8, so we need to use following methods to convert between the default and utf8 encoding for a given text to be used as system file path, such as `ParaIO.open(filename)` requires `filename` to be in default encoding.

```
NPL.load("(gl)script/ide/Encoding.lua");
local Encoding = commonlib.gettable("commonlib.Encoding");
commonlib.Encoding.Utf8ToDefault(text)
commonlib.Encoding.DefaultToUtf8(text)
```

Json Encoding

- Json Encoding: script/ide/Json.lua

```
NPL.load("(gl)script/ide/Json.lua");
local t = {
["name1"] = "value1",
["name2"] = {1, false, true, 23.54, "a \021 string"},
name3 = commonlib.Json.Null()
}

local json = commonlib.Json.Encode (t)
print (json)
--> {"name1":"value1","name3":null,"name2":[1,false,true,23.54,"a \u0015 string"]}

local t = commonlib.Json.Decode(json)
print(t.name2[4])
--> 23.54

-- also consider the NPL version
local out={};
if(NPL.FromJson(json, out)) then
    commonlib.echo(out)
end
```

XML Encoding

- XML Encoding: script/ide/LuaXML.lua

```
NPL.load("(gl)script/ide/LuaXML.lua");
function TestLuaXML:test_LuaXML_CPlusPlus()
    local input = [[<paragraph justify="centered" >first child<b >bold</b>second child
↪</paragraph>]]
    local x = ParaXML.LuaXML_ParseString(input);
    assert(x[1].name == "paragraph");
end

function TestLuaXML:test_LuaXML_NPL()
    local input = [[<paragraph justify="centered" >first child<b >bold</b>second child
↪</paragraph>]]
    local xmlRoot = commonlib.XML2Lua(input)
    assert(commonlib.Lua2XmlString(xmlRoot) == input);
    log(commonlib.Lua2XmlString(xmlRoot, true))
end
```

Binary Data

To read or write binary data to or from string, we can use the file API with a special filename called `<memory>`, see below.

- To read binary string, simply call `WritingString` to write input string to a memory buffer file and then `seek(0)` and read data out in any way you like.
- To write binary string, simply call `WritingString`, `WriteBytes` or `WritingInt`, once finished, call `GetText(0, -1)` to get the final output binary string.

```
function test_MemoryFile()
    -- "<memory>" is a special name for memory file, both read/write is possible.
    local file = ParaIO.open("<memory>", "w");
    if(file:IsValid()) then
        file:WriteString("hello ");
        local nPos = file:GetFileSize();
        file:WriteString("world");
        file:WriteInt(1234);
        file:seek(nPos);
        file:WriteString("World");
        file:SetFilePointer(0, 2); -- 2 is relative to end of file
        file:WriteInt(0);
        file:WriteString("End");
        file:WriteBytes(3, {100, 0, 22});
        -- read entire binary text data back to npl string
        echo(#(file:GetText(0, -1)));
        file:close();
    end
end
```

XPath query in XML

- XPath: script/ide/XPath.lua

```
NPL.load("(gl)script/ide/XPath.lua");
```

```
local xmlDocIP = ParaXML.LuaXML_ParseFile("script/apps/Poke/IP.xml");
local xpath = "/mcml:mcml/mcml:packageList/mcml:package/";
local xpath = "//mcml:IPList/mcml:IP[@text = 'Level2_2']";
local xpath = "//mcml:IPList/mcml:IP[@version = 5]";
local xpath = "//mcml:IPList/mcml:IP[@version < 6]"; -- only supported by selectNodes2
local xpath = "//mcml:IPList/mcml:IP[@version > 4]"; -- only supported by selectNodes2
local xpath = "//mcml:IPList/mcml:IP[@version >= 5]"; -- only supported by␣
↪selectNodes2
local xpath = "//mcml:IPList/mcml:IP[@version <= 5]"; -- only supported by␣
↪selectNodes2

local xmlDocIP = ParaXML.LuaXML_ParseFile("character/v3/Pet/MGBB/mgbb.xml");
local xpath = "/mesh/shader/@index";
local xpath = "/mesh/boundingbox/@minx";
local xpath = "/mesh/submesh/@filename";
local xpath = "/mesh/submesh";

--
-- select nodes to an array table
--
local result = commonlib.XPath.selectNodes(xmlDocIP, xpath);
local result = XPath.selectNodes(xmlDocIP, xpath, nMaxResultCount); -- select at most␣
↪nMaxResultCount result

--
-- select a single node or nil
--
local node = XPath.selectNode(xmlDocIP, xpath);

--
-- iterate on all nodes.
--
for node in commonlib.XPath.eachNode(xmlDocIP, xpath) do
    commonlib.echo(node[1]);
end
```

Logging

log.txt may take a few seconds to be flushed to disk file.

- Logging: script/ide/log.lua

```
-- write formatted logs to log.txt
LOG.std(nil,"info", "sub_system_name", "some formated message: %s", "hello");
LOG.std(nil,"debug", "sub_system_name", {a=1, c="any table object"});
LOG.std(nil,"error", "sub_system_name", "error code here");
LOG.std(nil,"warn", "sub_system_name", "warning");
```

Log redirection and configurations: It is also possible to redirect log.txt to different files on startup. see [[NPLCommandLine]]

```
NPL.load("(gl)script/ide/log.lua");
commonlib.log("hello %s \n", "paraengine")
commonlib.log({"anything"})
local fromPos = commonlib.log.GetLogPos()
log(fromPos.." babababa...\n");
```

```
local text = commonlib.log.GetLog(fromPos, nil)
log(tostring(text).." is retrieved\n")

commonlib.applog("hello paraengine"); --> ./log.txt --> 20090711 02:59:19|0|hello␣
↪paraengine|script/shell_loop.lua:23: in function FunctionName|
commonlib.applog("hello %s", "paraengine")

commonlib.servicelog("MyService", "hello paraengine"); --> ./MyService_20090711.log --
↪> 2009-07-11 10:53:27|0|hello paraengine||
commonlib.servicelog("MyService", "hello %s", "paraengine");

-- set log properties before using the log
commonlib.servicelog.GetLogger("no_append"):SetLogFile("log/no_append.log")
commonlib.servicelog.GetLogger("no_append"):SetAppendMode(false);
commonlib.servicelog.GetLogger("no_append"):SetForceFlush(true);
commonlib.servicelog("no_append", "test");

-- This will change the default logger's file position at runtime.
commonlib.servicelog.GetLogger(""):SetLogFile("log/log_2016.5.19.txt");
```

CHAPTER 37

HTTP request

http url request: script/ide/System/os/GetUrl.lua

```
-- return the content of a given url.
-- e.g.  echo(NPL.GetURL("www.paraengine.com"))
-- @param url: url string or a options table of {url=string, postfields=string, form=
↪{key=value}, headers={key=value, "line strings"}, json=bool, qs={}}
-- if .json is true, code will be decoded as json.
-- if .qs is query string table
-- if .postfields is a binary string to be passed in the request body. If this is␣
↪present, form parameter will be ignored.
-- @param callbackFunc: a function(rcode, msg, data) end, if nil, the function will␣
↪not return until result is returned(sync call).
--  `rcode` is http return code, such as 200 for success, which is same as `msg.rcode`
--  `msg` is the raw HTTP message {header, code=0, rcode=200, data}
--  `data` contains the translated response data if data format is a known format␣
↪like json
--    or it contains the binary response body from server, which is same as `msg.data`
-- @param option: mostly nil. "-I" for headers only
-- @return: return nil if callbackFunc is a function. or the string content in sync␣
↪call.
function System.os.GetUrl(url, callbackFunc, option)
end
```

```
NPL.load("(gl)script/ide/System/os/GetUrl.lua");
```

- download file or making standard request

```
System.os.GetUrl("https://github.com/LiXizhi/HourOfCode/archive/master.zip", echo);`
```

- get headers only with "-I" option.

```
System.os.GetUrl("https://github.com/LiXizhi/HourOfCode/archive/master.zip",␣
↪function(err, msg, data)  echo(msg) end, "-I");
```

- send form KV pairs with http post

```
System.os.GetUrl({url = "http://localhost:8099/ajax/console?action=getparams", form =
↪{key="value",} }, function(err, msg, data)        echo(data)   end);
```

- send multi-part binary forms with http post

```
System.os.GetUrl({url = "http://localhost:8099/ajax/console?action=printrequest",␣
↪form = {name = {file="dummy.html",    data="<html><bold>bold</bold></html>", type=
↪"text/html"}, } }, function(err, msg, data)     echo(data)   end);
```

- To send any binary data, one can use

```
System.os.GetUrl({url = "http://localhost:8099/ajax/console?action=printrequest",␣
↪headers={["content-type"]="application/json"}, postfields='{"key":"value"}' },␣
↪function(err, msg, data)        echo(data)   end);
```

- To simplify json encoding, we can send form as json string using following shortcut

```
System.os.GetUrl({url = "http://localhost:8099/ajax/console?action=getparams", json =␣
↪true, form = {key="value", key2 ={subtable="subvalue"} } }, function(err, msg,␣
↪data)      echo(data)  end);
```

HTTP PUT request:

```
System.os.GetUrl({
    method = "PUT",
    url = "http://localhost:8099/ajax/log?action=log",
    form = {filecontent = "binary string here", }
}, function(err, msg, data)     echo(data)   end);
```

HTTP DELETE request:

```
System.os.GetUrl({
    method = "DELETE",
    url = "http://localhost:8099/ajax/log?action=log",
    form = {filecontent = "binary string here", }
}, function(err, msg, data)     echo(data)   end);
```

On the server side, suppose it is NPL web server, one can get the request body using `request:GetBody()` or `request:getparams()`. The former will only include request body, while the latter contains url parameters as well.

Debugging HTTP request

In NPL Code Wiki's console window, one can test raw http request by sending a request to `/ajax/console?action=printrequest`, and then check log console for raw request content.

```
System.os.GetUrl({url = "http://localhost:8099/ajax/console?action=printrequest",␣
↪echo);
```

Local Server

Local server offers a way for you to download files or making url requests with a cache policy. Local database system is used to backup all url requests. The cached files may be in the database or in native file system.

For more information, please see localserver

what is a local server?

The LocalServer module allows a web application to cache and serve its HTTP resources locally, without a network connection.

Local Server Overview

The LocalServer module is a specialized URL cache that the web application controls. Requests for URLs in the LocalServer's cache are intercepted and served locally from the user's disk.

Resource stores

A resource store is a container of URLs. Using the LocalServer module, applications can create any number of resource stores, and a resource store can contain any number of URLs.

There are two types of resource stores: - ResourceStore - for capturing ad-hoc URLs using NPL. The ResourceStore allows an application to capture user data files that need to be addressed with a URL, such as a PDF file or an image. - ManagedResourceStore - for capturing a related set of URLs that are declared in a manifest file, and are updated automatically. The ManagedResourceStore allows the set of resources needed to run a web application to be captured. For both types of stores, the set of URLs captured is explicitly controlled by the web application.

Architecture & Implementation Notes

all sql database manipulation functions are exposed via WebCacheDB, whose implementation is split in WebCacheDB* files. localserver is the based class for two servers: ResourceStore and ManagedResourceStore.

Using Local server as a local database

One can use local server as a simple (name, value) pair database with cache_policy functions.

To query a database entry call below, here we will use web service store

```
    NPL.load("(gl)script/ide/System/localserver/factory.lua");
    local ls = System.localserver.CreateStore(nil, 2);
    if(not ls) then
        return
    end
    cache_policy = cache_policy or System.localserver.CachePolicy:new("access plus 1␣
↪week");

    local url = System.localserver.UrlHelper.WS_to_REST(fakeurl_query_miniprofile,
↪{JID=JID}, {"JID"});
    local item = ls:GetItem(url)
    if(item and item.entry and item.payload and not cache_policy:IsExpired(item.
↪payload.creation_date)) then
        -- NOTE:item.payload.data is always a string, one may deserialize from it to␣
↪obtain table object.
        local profile = item.payload.data;
        if(type(callbackFunc) == "function") then
            callbackFunc(JID, profile);
        end
    else

    end
```

To add(update) a database entry call below

```
    NPL.load("(gl)script/ide/System/localserver/factory.lua");
    local ls = System.localserver.CreateStore(nil, 2);
    if(not ls) then
        return
    end
    -- make url
    local url = System.localserver.UrlHelper.WS_to_REST(fakeurl_query_miniprofile,
↪{JID=JID}, {"JID"});

    -- make entry
    local item = {
        entry = System.localserver.WebCacheDB.EntryInfo:new({
            url = url,
        }),
        payload = System.localserver.WebCacheDB.PayloadInfo:new({
            status_code = System.localserver.HttpConstants.HTTP_OK,
            data = msg.profile,
        }),
    }
    -- save to database entry
    local res = ls:PutItem(item)
    if(res) then
        log("ls put JID mini profile for "..url.."\n")
    else
        log("warning: failed saving JID profile item to local server.\n")
    end
```

Lazy writing

For the URL history, this transaction commit overhead is unacceptably high(0.05s for the most simple write commit). On some systems, the cost of committing a new page to the history database was as high as downloading the entire page and rendering the page to the screen. As a result, ParaEngine's localserver has implemented a lazy sync system.

Please see https://developer.mozilla.org/en/Storage/Performance, for a reference

Localserver has relaxed the ACID requirements in order to speed up commits. In particular, we have dropped durability. This means that when a commit returns, you are not guaranteed that the commit has gone through. If the power goes out right away, that commit may (or may not) be lost. However, we still support the other (ACI) requirements. This means that the database will not get corrupted. If the power goes out immediately after a commit, the transaction will be like it was rolled back: the database will still be in a consistent state.

Send Email Via SMTP

It is not trivial to make a SMTP email server. However, it is fairly easy to send an email via an existing email server. One can do it manually via `System.os.GetUrl` with proper options, or we write the following handy function for you.

```
System.os.SendEmail({
    url="smtp://smtp.exmail.qq.com",
    username="lixizhi@paraengine.com", password="XXXXX",
    -- ca_info = "/path/to/certificate.pem",
    from="lixizhi@paraengine.com", to="lixizhi@yeah.net", cc="xizhi.li@gmail.com",
    subject = "title here",
    body = "any body context here. can be very long",
}, function(err, msg) echo(msg) end);
```

Here is what I receive in my mail box:

CHAPTER 38

Files API

Search Files

- Files: script/ide/Files.lua

```
NPL.load("(gl)script/ide/Files.lua");
local result = commonlib.Files.Find({}, "model/test", 0, 500, function(item)
    local ext = commonlib.Files.GetFileExtension(item.filename);
    if(ext) then
        return (ext == "x") or (ext == "dds")
    end
end)

-- search zip files using perl regular expression. like ":^xyz\\s+.*blah$"
local result = commonlib.Files.Find({}, "model/test", 0, 500, ":.*", "*.zip")

-- using lua file system
local lfs = commonlib.Files.GetLuaFileSystem();
echo(lfs.attributes("config/config.txt", "mode"))
```

Read/Write Binary Files

```
    local file = ParaIO.open("temp/binaryfile.bin", "w");
    if(file:IsValid()) then
        local data = "binary\0\0\0\0file";
        file:WriteString(data, #data);
        -- write 32 bits int
        file:WriteUInt(0xffffffff);
        file:WriteInt(-1);
        -- write float
        file:WriteFloat(-3.14);
        -- write double (precision is limited by lua double)
        file:WriteDouble(-3.1415926535897926);
        -- write 16bits word
        file:WriteWord(0xff00);
        -- write 16bits short integer
        file:WriteShort(-1);
        file:WriteBytes(3, {255, 0, 255});
        file:close();

        -- testing by reading file content back
        local file = ParaIO.open("temp/binaryfile.bin", "r");
        if(file:IsValid()) then
            -- test reading binary string without increasing the file cursor
            assert(file:GetText(0, #data) == data);
            file:seekRelative(#data);
            assert(file:getpos() == #data);
            file:seek(0);
            -- test reading binary string
            assert(file:ReadString(#data) == data);
            assert(file:ReadUInt() == 0xffffffff);
            assert(file:ReadInt() == -1);
            assert(math.abs(file:ReadFloat() - (-3.14)) < 0.000001);
            assert(file:ReadDouble() == -3.1415926535897926);
            assert(file:ReadWord() == 0xff00);
            assert(file:ReadShort() == -1);
            local o = {};
            file:ReadBytes(3, o);
            assert(o[1] == 255 and o[2] == 0 and o[3] == 255);
            file:seek(0);
            assert(file:ReadString(8) == "binary\0\0");
            file:close();
        end
    end
```

In-Memory File And Binary Buffer

To write binary data to string, we can use the file API with a special filename called `<memory>`, see below

```
function test_MemoryFile()
    -- "<memory>" is a special name for memory file, both read/write is possible.
    local file = ParaIO.open("<memory>", "w");
    if(file:IsValid()) then
        file:WriteString("hello ");
        local nPos = file:GetFileSize();
        file:WriteString("world");
        file:WriteInt(1234);
        file:seek(nPos);
        file:WriteString("World");
        file:SetFilePointer(0, 2); -- 2 is relative to end of file
        file:WriteInt(0);
        file:WriteString("End");
        file:WriteBytes(3, {100, 0, 22});
        -- read entire binary text data back to npl string
        echo(#(file:GetText(0, -1)));
        file:close();
    end
```

```
end
```

Read/Write Files

see `NPL.load("(gl)script/test/ParaIO_test.lua");`

```
-- tested on 2007.1, LiXizhi
local function ParaIO_FileTest()

    -- file write
    log("testing file write...\r\n")

    local file = ParaIO.open("temp/iotest.txt", "w");
    file:WriteString("test\r\n");
    file:WriteString("test\r\n");
    file:close();

    -- file read
    log("testing file read...\r\n")

    local file = ParaIO.open("temp/iotest.txt", "r");
    log(tostring(file:readline()));
    log(tostring(file:readline()));
    log(tostring(file:readline()));
    file:close();

end

-- tested on 2007.6.7, LiXizhi
local function ParaIO_ZipFileTest()
    local writer = ParaIO.CreateZip("d:\\simple.zip","");
    writer:ZipAdd("temp/file1.ini", "d:\\file1.ini");
    writer:ZipAdd("temp/file2.ini", "d:\\file2.ini");
    writer:ZipAddFolder("temp");
    writer:AddDirectory("worlds/", "d:/temp/*.", 4);
    writer:AddDirectory("worlds/", "d:/worlds/*.*", 2);
    writer:close();
end

-- tested on 2007.6.7, LiXizhi
local function ParaIO_SearchZipContentTest()
    -- test case 1
    log("test case 1\n");
    local search_result = ParaIO.SearchFiles("","*.", "d:\\simple.zip", 0, 10000, 0);
    local nCount = search_result:GetNumOfResult();
    local i;
    for i = 0, nCount-1 do
        log(search_result:GetItem(i).."\n");
    end
    search_result:Release();
    -- test case 2
    log("test case 2\n");
    local search_result = ParaIO.SearchFiles("","*.ini", "d:\\simple.zip", 0, 10000,␣
↪0);
    local nCount = search_result:GetNumOfResult();
    local i;
    for i = 0, nCount-1 do
        log(search_result:GetItem(i).."\n");
    end
    search_result:Release();
    -- test case 3
    log("test case 3\n");
    local search_result = ParaIO.SearchFiles("","temp/*.", "d:\\simple.zip", 0, 10000,
↪ 0);
    local nCount = search_result:GetNumOfResult();
    local i;
    for i = 0, nCount-1 do
        log(search_result:GetItem(i).."\n");
    end
    search_result:Release();
    -- test case 4
    log("test case 4\n");
    local search_result = ParaIO.SearchFiles("temp/","*.*", "d:\\simple.zip", 0,␣
↪10000, 0);
    local nCount = search_result:GetNumOfResult();
    local i;
    for i = 0, nCount-1 do
        log(search_result:GetItem(i).."\n");
    end
    search_result:Release();
    -- test case 5
    log("test case 5\n");
    local search_result = ParaIO.SearchFiles("","temp/*.*", "d:\\simple.zip", 0,␣
↪10000, 0);
    local nCount = search_result:GetNumOfResult();
    local i;
    for i = 0, nCount-1 do
        log(search_result:GetItem(i).."\n");
    end
    search_result:Release();
end

local function ParaIO_SearchPathTest()
    ParaIO.CreateDirectory("npl_packages/test/")
    local file = ParaIO.open("npl_packages/test/test_searchpath.lua", "w");
    file:WriteString("echo('from test_searchpath.lua')")
    file:close();

    ParaIO.AddSearchPath("npl_packages/test");
    ParaIO.AddSearchPath("npl_packages/test/"); -- same as above, check for duplicate

    assert(ParaIO.DoesFileExist("test_searchpath.lua"));

    ParaIO.RemoveSearchPath("npl_packages/test");

    assert(not ParaIO.DoesFileExist("test_searchpath.lua"));

    -- ParaIO.AddSearchPath("npl_packages/test/");
    -- this is another way of ParaIO.AddSearchPath, except that it will check for␣
↪folder existence.
    -- in a number of locations.
    NPL.load("npl_packages/test/");

    -- test standard open api
    local file = ParaIO.open("test_searchpath.lua", "r");
    if(file:IsValid()) then
        log(tostring(file:readline()));
    else
        log("not found\n");
    end
    file:close();

    -- test script file
    NPL.load("(gl)test_searchpath.lua");

    ParaIO.ClearAllSearchPath();

    assert(not ParaIO.DoesFileExist("test_searchpath.lua"));
end

-- TODO: test passed on 2008.4.20, LiXizhi
function ParaIO_PathReplaceable()
    ParaIO.AddPathVariable("WORLD", "worlds/MyWorld")
    if(ParaIO.AddPathVariable("userid", "temp/LIXIZHI_PARAENGINE")) then
        local fullpath;
        commonlib.echo("test simple");
        fullpath = ParaIO.DecodePath("%WORLD%/%userid%/filename");
        commonlib.echo(fullpath);
        fullpath = ParaIO.EncodePath(fullpath)
        commonlib.echo(fullpath);

        commonlib.echo("test encoding with a specified variables");
        fullpath = ParaIO.DecodePath("%WORLD%/%userid%/filename");
        commonlib.echo(fullpath);
        commonlib.echo(ParaIO.EncodePath(fullpath, "WORLD"));
        commonlib.echo(ParaIO.EncodePath(fullpath, "WORLD, userid"));

        commonlib.echo("test encoding with inline path");
        fullpath = ParaIO.DecodePath("%WORLD%/%userid%_filename");
        commonlib.echo(fullpath);
        fullpath = ParaIO.EncodePath(fullpath)
        commonlib.echo(fullpath);


        commonlib.echo("test nested");
        fullpath = ParaIO.DecodePath("%userid%/filename/%userid%/nestedtest");
        commonlib.echo(fullpath);
        fullpath = ParaIO.EncodePath(fullpath)
        commonlib.echo(fullpath);

        commonlib.echo("test remove");
        if(ParaIO.AddPathVariable("userid", nil)) then
            fullpath = ParaIO.DecodePath("%userid%/filename");
            commonlib.echo(fullpath);
            fullpath = ParaIO.EncodePath(fullpath)
            commonlib.echo(fullpath);
        end

        commonlib.echo("test full path");
        fullpath = ParaIO.DecodePath("NormalPath/filename");
        commonlib.echo(fullpath);
        fullpath = ParaIO.EncodePath(fullpath)
        commonlib.echo(fullpath);
    end
end

function ParaIO_SearchFiles_reg_expr()
    -- test case 1
    local search_result = ParaIO.SearchFiles("script/ide/",":.*", "*.zip", 2, 10000,␣
↪0);
    local nCount = search_result:GetNumOfResult();
    local i;
    for i = 0, nCount-1 do
        log(search_result:GetItem(i).."\n");
    end
    search_result:Release();
end

function test_excel_doc_reader()
    NPL.load("(gl)script/ide/Document/ExcelDocReader.lua");
    local ExcelDocReader = commonlib.gettable("commonlib.io.ExcelDocReader");
    local reader = ExcelDocReader:new();

    -- schema is optional, which can change the row's keyname to the defined value.
    reader:SetSchema({
        [1] = {name="npcid", type="number"},
        [2] = {name="superclass", validate_func=function(value)  return value or
↪"menu1"; end },
        [3] = {name="class", validate_func=function(value)  return value or "normal";␣
↪end },
        [4] = {name="class_name", validate_func=function(value)  return value or "";␣
↪end },
        [5] = {name="gsid", type="number" },
        [6] = {name="exid", type="number" },
        [7] = {name="money_list", },
    })
    -- read from the second row
    if(reader:LoadFile("config/Aries/NPCShop/npcshop.xml", 2)) then
        local rows = reader:GetRows();
        echo(rows);
    end



    NPL.load("(gl)script/ide/Document/ExcelDocReader.lua");
    local ExcelDocReader = commonlib.gettable("commonlib.io.ExcelDocReader");
    local reader = ExcelDocReader:new();

    -- schema is optional, which can change the row's keyname to the defined value.
    local function card_and_level_func(value)
        if(value) then
            local level, card_gsid = value:match("^(%d+):(%d+)");
            return {level=level, card_gsid=card_gsid};
        end
    end
    reader:SetSchema({
        {name="gsid", type="number"},
        {name="displayname"},
        {name="max_level", type="number" },
        {name="hp", type="number" },
        {name="attack", type="number" },
        {name="defense", type="number" },
        {name="powerpips_rate", type="number" },
        {name="accuracy", type="number" },
        {name="critical_attack", type="number" },
        {name="critical_block", type="number" },
        {name="card1",  validate_func= card_and_level_func},
        {name="card2",  validate_func= card_and_level_func},
        {name="card3",  validate_func= card_and_level_func},
        {name="card4",  validate_func= card_and_level_func},
        {name="card5",  validate_func= card_and_level_func},
        {name="card6",  validate_func= card_and_level_func},
        {name="card7",  validate_func= card_and_level_func},
        {name="card8",  validate_func= card_and_level_func},
    })
    -- read from the second row
    if(reader:LoadFile("config/Aries/Others/combatpet_levels.excel.teen.xml", 2)) then
        local rows = reader:GetRows();
        log(commonlib.serialize(rows, true));
    end
end
```

Delete Files

See ParaIO Reference for more functions

```
ParaIO.DeleteFile("temp/*.*");
```

Compress and Decompress with Zlib and Gzip

- Files: script/test/TestNPL.lua

```
-- compress/decompress test
function TestNPL.Compress()
    -- using gzip
    local content = "abc";
    local dataIO = {content=content, method="gzip"};
    if(NPL.Compress(dataIO)) then
        echo(dataIO);
        if(dataIO.result) then
            dataIO.content = dataIO.result; dataIO.result = nil;
            if(NPL.Decompress(dataIO)) then
                echo(dataIO);
                assert(dataIO.result == content);
            end
        end
    end

    -- using zlib and deflate
    local content = "abc";
    local dataIO = {content=content, method="zlib", windowBits=-15, level=3};
    if(NPL.Compress(dataIO)) then
        echo(dataIO);
        if(dataIO.result) then
            dataIO.content = dataIO.result; dataIO.result = nil;
            if(NPL.Decompress(dataIO)) then
                echo(dataIO);
                assert(dataIO.result == content);
            end
        end
    end
end
```

Running Shell Commands

- source: script/ide/System/os/run.lua

To run external command lines via windows batch or linux bash shell in either `synchronous` or `asynchronous` mode.

```
NPL.load("(gl)script/ide/System/os/run.lua");
if(System.os.GetPlatform()=="win32") then
    -- any lines of windows batch commands
    echo(System.os("dir *.exe \n svn info"));
    -- this will popup confirmation window, so there is no way to get its result.
    System.os.runAsAdmin('reg add "HKCR\\paracraft" /ve /d "URL:paracraft" /f');
else
    -- any lines of linux bash shell commands
    echo(System.os.run("ls -al | grep total\ngit | grep commit"));
end
-- async run command in worker thread
for i=1, 10 do
    System.os.runAsync("echo hello", function(err, result)   echo(result)  end);
end
echo("waiting run async reply ...")
```

please note:

- windows and linux have different default shell program. One may need to use `System.os.GetPlatform()=="win32"` to target code to a given platform.
- We can write very long multi-line commands. Internally a temporary shell script file is created and executed with IO redirection.
- Sometimes, we may prefer async API to prevent stalling the calling thread, such as NPL web server invoking some backend programs for image processing. The async API pushes a message to a processor queue and returns immediately, one can use one or more NPL threads to process any number of queued shell commands. For more details, please see doc in `System.os.runAsync` source file.
- You need to have permissions to run these scripts. Under window 10 or above, we provide `System.os.runAdmin` to automatically popup confirmation dialog to run as administrator. Under linux, there is nothing we can do, you must give execute permission to `./temp` folder. This is the case for `selinux` core.

Reading Image File

```
function test_reading_image_file()
    -- reading binary image file
    -- png, jpg format are supported.
    local filename = "Texture/alphadot.png";
    local file = ParaIO.open(filename, "image");
    if(file:IsValid()) then
        local ver = file:ReadInt();
        local width = file:ReadInt();
        local height = file:ReadInt();
        -- how many bytes per pixel, usually 1, 3 or 4
        local bytesPerPixel = file:ReadInt();
        echo({ver, width=width, height = height, bytesPerPixel = bytesPerPixel})
        local pixel = {};
        for y=1, height do
            for x=1, width do
                pixel = file:ReadBytes(bytesPerPixel, pixel);
                echo({x, y, rgb=pixel})
            end
        end
        file:close();
    end
end
```

CHAPTER 39

Mouse and Key Input

This section is about handling mouse and key events in a window application.

Low level event handler

At the lowest level of NPL, mouse and keyboard event handlers can be registered to `ParaUIObject` and the global `ParaScene`. The ParaScene is a global singleton to handle all events not handled by any GUI objects. Please note, there is a another `AutoCameraController` which is enabled by default to handle mouse and key events before passing to ParaScene. So the order of event filtering in NPLRuntime is like below

- GUI events: controls that have focus always get event first
- `AutoCameraController`: optionally enabled for handling basic `player control`, such as right click to rotate the view, arrow keys to move the main player. Please note, this can be disabled if one wants to handle everything in NPL script. Trust me, it can cost you over 2000 lines of code if you do it manually.
- ParaScene: finally mouse and key events are handled by the 3d scene.

It is NOT recommended to use low level NPL api directly, see next section.

Use NPL libraries to handle event

2D events

For 2D GUI events, one can handle via window controls or MCML page. There is also a mcml v1 tag called `<pe:hotkey>` for handling simple key event when window is visible.

3D Scene Context

For 3D or global mouse/key, one should use SceneContext.

At most one scene context can be selected at any time. Once selected, the scene context will receive all key/mouse events in the 3D scene. One can derive from this class to write and switch to your own scene event handlers.

The computational model for global key/mouse event in C++ Engine is:

```
    key/mouse input--> 2D --> (optional auto camera) --> 3D scene --> script handlers
```

This class simplified and modified above model with following object oriented model:

```
    key/mouse input--> 2D --> one of SceneContext object--> Manipulator container -->␣
↪ Manipulators --> (optional auto camera manipulator)
```

Please note that both model can coexist, however SceneContext is now the recommended way to handle any scene event. SceneContext hide all dirty work of hooking into the old C++ engine callback interface, and offers more user friendly way of event handling.

Virtual functions:

```
    mousePressEvent(event)
    mouseMoveEvent
    mouseReleaseEvent
    mouseWheelEvent
    keyReleaseEvent
    keyPressEvent
    OnSelect()
    OnUnselect()
```

use the lib:

```
NPL.load("(gl)script/ide/System/Core/SceneContext.lua");
local MySceneContext = commonlib.inherit(commonlib.gettable("System.Core.SceneContext
↪"), commonlib.gettable("System.Core.MySceneContext"));
function MySceneContext:ctor()
    self:EnableAutoCamera(true);
end

function MySceneContext:mouseReleaseEvent(event)
    _guihelper.MessageBox("clicked")
end

-- method 1:
local sContext = MySceneContext:new():Register("MyDefaultSceneContext");
sContext:activate();
-- method 2:
MySceneContext:CreateGetInstance("MyDefaultSceneContext"):activate();
```

Scene Context Switch

Context is usually associated with a tool item, such as when user click to use the tool, its context is activated and once user deselect it, the context is switched back to a default one.

Scene context allows one to write modular code for each tool items with complex mouse/key input without affecting each other. It is common for an application to derive all of its context from a base context to support shared global key events, etc.

For an example of using context, please see Paracraft's context folder.

See also here for how to replace the default context in paracraft's mod interface with filters.

Manipulators

Scene context can contain manipulators. Manipulator is a module to manipulate complex scene objects, like the one below.

Manipulator is the base class used for creating user-defined manipulators. A manipulator can be connected to a depend node instead of updating a node attribute directly call AddValue() in constructor if one wants to define a custom manipulator property(plug) that can be easily binded with dependent node's plug.

Manipulator can also draw complex 3D overlay objects with 3d picking support. In above picture, the three curves are drawn by the Rotate manipulator.

CHAPTER 40

Overview

To write modular code, a system may expose its interface in the form of virtual functions, events or filters.

- `virtual functions`: allows a derived class to hook the input and output of a single function in the base class.
- `events`: feeds input to external functions when certain things happened. Events only hooks the input without providing an output.
- `filters`: allows external functions to form input-output chains to modify data at any point of execution.

Use whatever patterns to allow external modifications to the input-output of your modular code.

Filters

Filters is a design pattern of input-output chains. Please see script/ide/System/Core/Filters.lua for detailed usage.

You can simply apply a named filters at any point of execution in your code to allow other users to modify your data.

For example, one can turn `data = data;` into an equivalent filter, like below

```
data = GameLogic.GetFilters():apply_filters("my_data", data, some_parameters);
```

The above code does nothing until some other modules `add_filters` to `"my_data"`.

Each filter function takes the output of the previous filter function as its first parameter, all other input parameters are shared by all filter functions, like below

```
F1(input, ...)-->F2(F1_output, ...)-->F3(F2_output, ...)-->F3_output
```

Filters is the recommended way for plugin/app developers to extend or modify paracraft.

Click here to see all Paracraft Filters

CHAPTER 41

Localization

There are some helper class for you to implement localization. For best practices, please see example in paracraft package, which uses `poedit` to edit and store localization strings with multi-languages.

- Helper class for translation table:
  - script/ide/Locale.lua
- paracraft examples:
  - Translation
  - TranslationGetText

Overview

UTF8 encoding should be used where ever possible in your source code or mcml files. Everything returned from NPL is also UTF8 encoded, except for native file path.

Follow following steps:

- Use this class to create a global table `L` to look up for localized text with a primary key.
- When your application start, populate `table L` with data from your localization files according to current language setting.
- In your script code or mcml UI page files, replace any text "XXX" with `L"XXX"`
- Re-Run `Poedit` to scan for all source files and update the localization database file.
- Inform your translator to translate the `*.po` files into multiple languages and generate `*.mo` files. Or you may just use google translate or other services.
- Iterate above three steps when you insert new text in your source code.

  Always use

```
-- Right way
local text = format(L"Some text %d times", times)
```

instead of

```
-- BAD & Wrong Way!!!
local text = L"Some text "..times..L" times"
```

for better translation because different language have different syntax.

Localization GetText Tools

To store your language strings, you can use plain table in script or use a third-party tool like Poedit. We have created `Poedit` plugin to support NPL code, download the NPLgettext tool here.

In paracraft, there is a command called `/poedit` which does the `gettext` job automatically.

CHAPTER 42

User Interface

There are two low-level ways to draw 2d graphical objects.

- One is via creating ParaUIObject controls (like buttons, containers, editbox). These controls are managed in C++ and created in NPL scripts.
- The other is creating owner draw ParaUIObject, and do all the drawings with Painting API in NPL scripts, such as draw rect, lines, text, etc.

Moreover, there are a number of high-level graphical libraries written in NPL which allows you to create 2D user interface more easily.

- IDE controls is NPL wrapper of the low-level C++ API. It provides more versatile controls than the raw ParaUIObjects.
  - One old implementation is based on ParaUIObject controls.
  - One new implementation is based on Painting API, in `System.Window.UIElement` namespace.
- MCML/NPL is a HTML/JS like mark up language to create user interface.
  - One old implementation is based on ParaUIObject controls.
  - One new implementation is based on Painting API, in `System.Window.mcml` namespace.

Further Reading

- [[Low-level drawing API | DrawingAPI]]
- [[UI Controls | System.Window]]
- [[mcml]]

CHAPTER 43

Drawing With 2D API

There are two low-level ways to draw 2d graphical objects.

- One is via creating ParaUIObject controls (like buttons, containers, editbox). These controls are managed in C++ and created in NPL scripts.
- The other is creating owner draw ParaUIObject, and do all the drawings with Painting API in NPL scripts, such as draw rect, lines, text, etc.

Moreover, there is a number of high-level graphical libraries written in NPL which allows you to create 2D user interface more easily.

- IDE controls is NPL wrapper of the low-level C++ API. It provides more versatile controls than the raw ParaUIObjects.
  - One old implementation is based on ParaUIObject controls.
  - One new implementation is based on Painting API, in `System.Window.UIElement` namespace.
- MCML/NPL is a HTML/JS like mark up language to create user interface.
  - One old implementation is based on ParaUIObject controls.
  - One new implementation is based on Painting API, in `System.Window.mcml` namespace.

  This article is only about low-level drawing API in C++ side, which are exposed to NPL. However, one should always use MCML or Windows.UIElement instead of following raw API.

button

Creating a button with a texture background and text. The root element is a container see next section. Please place a png texture `Texture/alphadot.png` at the working directory.

```
NPL.load("(gl)script/ide/System/System.lua");

local clickCount = 1;
local function CreateUI()
```

```
   local _this = ParaUI.CreateUIObject("button", "MyBtn", "_lt", 10, 10, 64, 22);
   _this.text= "text";
   _this.background = "Texture/alphadot.png";
   _this:SetScript("onclick", function(obj)
      obj.text = "clicked "..clickCount;
      clickCount = clickCount + 1;
   end)
   _this:AttachToRoot();
end
CreateUI();
```

container

Create a container and a child button inside it.

```
    local _parent, _this;
    _this = ParaUI.CreateUIObject("container", "MyContainer", "_lt", 10, 110, 200,␣
↪64);
    _this:AttachToRoot();
    _this.background = "Texture/alphadot.png";
    _parent = _this;

    -- create a child
    _this = ParaUI.CreateUIObject("button", "b", "_rt", -10-32, 10, 32, 22);
    _this.text= "X"
    _this.background = "";
    _parent:AddChild(_this);
```

text and editbox

```
```

System Fonts

In ParaEngine/NPL, we support loading fonts from `*.ttf` files. One can install custom font files at `fonts/*.ttf`, and use them with filename, such as `Verdana;bold`. Please note, all controls in ParaEngine uses "System" font by default. "System" font maps to the system font used by the current computer by default. However, one can change it to a different custom font installed in `fonts/*.ttf`. Suppose you have a font file at `fonts/ParaEngineThaiFont.ttf`, you can map default System font to it by calling.

```
Config.AppendTextValue("GUI_font_mapping","System");
Config.AppendTextValue("GUI_font_mapping","ParaEngineThaiFont");
```

It is a pair of function calls, the first is name, the second is value. We recommend doing it in `script/config.lua`, it is loaded before any UI control is created. Please see that file for details.

Please note, for each font with different size, weight, and name, we will create a different internal temporary image file for rendering. So it is recommended to use just a few fonts in your application.

CHAPTER 44

2D GUI Windows

A `System.Windows.Window` is the primary container for 2D GUI. There are multiple ways to create windows. The most common way is to use `mcml` markup language. The programing model is like writing `HTML/js` web page.

Quick Sample

First you need to create a html file, such as `source/HelloWorldMCML/mcml_window.html`.

```
<pe:mcml>
<script refresh="false" type="text/npl" src="mcml_window.lua"><![CDATA[
    function OnClickOK()
        local content = Page:GetValue("content") or "";
        _guihelper.MessageBox(":"..content);
    end
]]></script>
<div style="color:#33ff33;background-color:#808080">
    <div style="margin:10px;">
        Hello World from HTML page!
    </div>
    <div style="margin:10px;">
        <input type="text" name="content" style="width:200px;height:25px;" />
        <input type="button" name="ok" value="" onclick="OnClickOK"/>
    </div>
</div>
</pe:mcml>
```

To create and show the window using the html file, we simply do following code.

```
    NPL.load("(gl)script/ide/System/Windows/Window.lua");
    local Window = commonlib.gettable("System.Windows.Window")
    local window = Window:new();
    window:Show({
        url="source/HelloWorldMCML/mcml_window.html",
        alignment="_lt", left = 300, top = 100, width = 300, height = 400,
    });
```

Window Alignment

Notice the alignment parameter specify the relative position to its parent or the native screen window.

```
        * @param alignment: can be one of the following strings or nil or left out␣
↪entirely:
        *   - "_lt" align to left top of the screen
        *   - "_lb" align to left bottom of the screen
        *   - "_ct" align to center of the screen
        *   - "_ctt": align to center top of the screen
        *   - "_ctb": align to center bottom of the screen
        *   - "_ctl": align to center left of the screen
        *   - "_ctr": align to center right of the screen
        *   - "_rt" align to right top of the screen
        *   - "_rb" align to right bottom of the screen
        *   - "_mt": align to middle top
        *   - "_ml": align to middle left
        *   - "_mr": align to middle right
        *   - "_mb": align to middle bottom
        *   - "_fi": align to left top and right bottom. This is like fill in the␣
↪parent window.
        *
        *   the layout is given below:
        *   _lt _mt _rt
        *   _ml _ct _mr
        *   _lb _mb _rb
```

`left, top, width, height` have different meanings for different alignment type. The most simple one is "_lt" which means left top alignment.

- center alignment:
  - `alignment="_ct", left = 0, top = -100, width = 300, height = 400` will display window at center. Normally, if you want to center a window with given `width` and `height`, one should use: `left = -width/2, top = -height/2, width, height`. The left, top means relative to the center of the parent window. "_ctt" means center top.
- middle alignment:
  - "_mt" middle top: it means that the pixels relative to left and right border of the parent window should be `fixed`.In other words, if the parent resizes, the width of the window also resize.
  - `_mt`: x is coordinate from the left. y is coordinate from the top, width is the coordinate from the right and height is the height
  - `_mb`: x is coordinate from the left. y is coordinate from the bottom, width is the coordinate from the right and height is the height
  - `_ml`: x is coordinate from the left. y is coordinate from the top, width is the width and height is the coordinate from the bottom
  - `_mr`: x is coordinate from the right. y is coordinate from the top, width is the width and height is the coordinate from the bottom

MCML V1 vs V2

There two implementations of MCML: `v1` and `v2`. The example above is given by v2, which is the preferred implementation. However, v1 is mature and stable implementation, v2 is new and currently does not have as many build-in controls as v1. We are still working on v2 and hope it can be 100% compatible and powerful with v1. To launch the same mcml page with v1, one can use

```
    NPL.load("(gl)script/kids/3DMapSystemApp/mcml/PageCtrl.lua");
    local page = System.mcml.PageCtrl:new({url="source/HelloWorldMCML/mcml_window.html
↪"});
    page:Create("testpage", nil, "_lt", 0, 0, 300, 400);
```

Difference between v1 and v2

- v1 uses NPL's build-in GUI objects written in C++, i.e. ParaUIObject like `button`, `container`, `editbox`. Their implementation is hidden from NPL scripts and user IO is exposed to NPL script via callback functions.
- v2 uses NPL script to draw all the GUI objects using 3d-triangle drawing API and handles user input in NPL script. Therefore the user has complete control over the look of their control in NPL script. One can read the source code of v2 in main package.

UI Elements (Controls)

In mcml v2 implementation, mcml page are actually translated to UI Elements at runtime. UI Element contains a collection of GUI controls written completely in NPL. They all derive from UIElement, which contains virtual functions to paint the control and handle user IO event.

The following are buildin-in UI elements or controls:

- Button
- EditBox
- Label
- Canvas
- Window

Creating GUI Without MCML

In most cases, we should use mcml to create GUI, however, there are places when you want to create your custom control or custom mcml tag, you will then need to write your own control.

Following are some examples:

```
function test_Windows:TestCreateWindow()

    -- create the native window
    local window = Window:new();

    -- test UI element
    local elem = UIElement:new():init(window);
    elem:SetBackgroundColor("#0000ff");
    elem:setGeometry(10,0,64,32);
    -- test create rectangle
    local rcRect = Rectangle:new():init(window);
    rcRect:SetBackgroundColor("#ff0000");
    rcRect:setGeometry(10,32,64,32);

    -- test Button
    local btn = Button:new():init(window);
    btn:SetBackgroundColor("#00ff00");
    btn:setGeometry(10,64,64,32);
    btn:Connect("clicked", function (event)
        _guihelper.MessageBox("you clicked me");
    end)
    btn:Connect("released", function(event)
        _guihelper.MessageBox("mouse up");
    end)

    -- show the window natively
    window:Show("my_window", nil, "_mt", 0,0, 200, 200);
end

function test_Windows:TestMouseEnterLeaveEvents()
    -- create the native window
    local window = Window:new();
    window.mouseEnterEvent = function(self, event)
        Application:postEvent(self, System.Core.LogEvent:new({"window enter",␣
↪event:localPos()}));
    end
    window.mouseLeaveEvent = function(self, event)
        Application:postEvent(self, System.Core.LogEvent:new({"window leave"}));
    end

    -- Parent1
    local elem = UIElement:new():init(window);
    elem:SetBackgroundColor("#0000ff");
    elem:setGeometry(10,0,64,64);
    elem.mouseEnterEvent = function(self, event)
        Application:postEvent(self, System.Core.LogEvent:new({"parent1 enter",␣
↪event:localPos()}));
    end
    elem.mouseLeaveEvent = function(self, event)
        Application:postEvent(self, System.Core.LogEvent:new({"parent1 leave"}));
    end

    -- Parent1:Button1
    local btn = Button:new():init(elem);
    btn:SetBackgroundColor("#ff0000");
    btn:setGeometry(0,0,64,32);
    btn.mouseEnterEvent = function(self, event)
        Application:postEvent(self, System.Core.LogEvent:new({"btn1 enter",␣
↪event:localPos()}));
    end
    btn.mouseLeaveEvent = function(self, event)
        Application:postEvent(self, System.Core.LogEvent:new({"btn1 leave"}));
    end

    -- Button2
    local btn = Button:new():init(window);
    btn:SetBackgroundColor("#00ff00");
    btn:setGeometry(10,64,64,32);
    btn.mouseEnterEvent = function(self, event)
        Application:postEvent(self, System.Core.LogEvent:new({"btn2 enter",␣
↪event:localPos()}));
    end
    btn.mouseLeaveEvent = function(self, event)
        Application:postEvent(self, System.Core.LogEvent:new({"btn2 leave"}));
    end

    -- show the window natively
    window:Show("my_window1", nil, "_mt", 0,200, 200, 200);
end

function test_Windows:TestEditbox()

    -- create the native window
    local window = Window:new();

    -- test UI element
    local elem = EditBox:new():init(window);
    elem:setGeometry(60,30,64,25);
    -- elem:setMaxLength(6);
    -- show the window natively
    window:Show("my_window", nil, "_lt", 0,0, 200, 200);
end
```

Creating Custom MCML v2 Controls

This is our custom control `MyApp.Controls.MyCustomControl`, whose implementation is defined in `test_pe_custom.lua`

```
<pe:mcml>
<script type="text/npl" refresh="false">
<![CDATA[
]]>
</script>
    <pe:custom src="test_pe_custom.lua" classns="MyApp.Controls.MyCustomControl"␣
↪style="width:200px;height:200px;">
    </pe:custom>
</pe:mcml>
```

Now in `test_pe_custom.lua`, we create its implementation.

```
local MyCustomControl = commonlib.inherit(commonlib.gettable("System.Windows.UIElement
↪"), commonlib.gettable("MyApp.Controls.MyCustomControl"));

function MyCustomControl:paintEvent(painter)
    painter:SetPen("#ff0000");
    painter:DrawText(10,10, "MyCustomControl");
end
```

Preview and Debug Layout

Sometimes, we want to change the code and preview the result without restarting the whole application. If your windows contains no external logics and uses mcml v1 exclusively, you can preview using MCML browser, simply press `Ctrl+F3`.

In most cases, you may need to write temporary or persistent test code for your new window class with data connected. This usually involves delete the old window object and create a new one like below. Run it repeatedly in NPL code wiki's console to preview your updated mcml code.

```
-- remove old window
local window = commonlib.gettable("test.window")
if(window and window.CloseWindow) then
    window:CloseWindow(true);
end

-- create a new window
NPL.load("(gl)script/ide/System/Windows/Window.lua");
local Window = commonlib.gettable("System.Windows.Window")
local window = Window:new();
window:Show({
    url="script/ide/System/test/test_mcml_page.html",
    alignment="_lt", left = 0, top = 0, width = 200, height = 400,
});
-- keep a reference for refresh
test.window = window;
```

CHAPTER 45

MCML Markup Language For 2D GUI

MCML or Micro Cosmos Markup Language is a meta language written in NPL since 2008, since then it has become the standard of writing 2D user interface in NPL. MCML is inspired by early version of ASP.net, however it uses local architecture for local user interface.

Quick Sample

Here is an example of mcml page in a `.html` file, such as `source/HelloWorldMCML/mcml_window.html`. The file extension only makes it possible for you to preview the page in other html code editor.

```
<pe:mcml>
<script refresh="false" type="text/npl" src="mcml_window.lua"><![CDATA[
    function OnClickOK()
        local content = Page:GetValue("content") or "";
        _guihelper.MessageBox(":"..content);
    end
]]></script>
<div style="color:#33ff33;background-color:#808080">
    <div style="margin:10px;">
        Hello World from HTML page!
    </div>
    <div style="margin:10px;">
        <input type="text" name="content" style="width:200px;height:25px;" />
        <input type="button" name="ok" value="" onclick="OnClickOK"/>
    </div>
</div>
</pe:mcml>
```

To render the page, one can launch it with a window like below. See [[System.Window]] for details.

```
    NPL.load("(gl)script/ide/System/Windows/Window.lua");
    local Window = commonlib.gettable("System.Windows.Window")
    local window = Window:new();
    window:Show({
        url="source/HelloWorldMCML/mcml_window.html",
        alignment="_lt", left = 300, top = 100, width = 300, height = 400,
    });
```

MCML V1 vs V2

There two implementations of MCML: `v1` and `v2`. The example above is given by v2, which is the preferred implementation. However, v1 is mature and stable implementation, v2 is new and currently does not have as many build-in controls as v1. We are still working on v2 and hope it can be 100% compatible and powerful with v1. This article is only about v2. See last section for v1.

Difference between v1 and v2

- v1 uses NPL's build-in GUI objects written in C++, i.e. ParaUIObject like `button`, `container`, `editbox`. Their implementation is hidden from NPL scripts and user IO is exposed to NPL script via callback functions.
- v2 uses NPL script to draw all the GUI objects using 3d-triangle drawing API and handles user input in NPL script. Therefore the user has complete control over the look of their control in NPL script. One can read the source code of v2 in main package.

Mcml tags

There are a number of tags which you can use to create interactive content. Besides, one can also create your own custom tags with UI controls, see [[System.Window]] for details. All mcml build-in tags are in the `pe:` xml namespace, which stands for ParaEngine or pe.

- pe:div
- pe:button
- pe:container
- pe:custom
- pe:editbox
- pe:identicon
- pe:if
- pe:input
- pe:repeat
- pe:script
- pe:span
- pe:text

Standard html tags are mapped as follows:

- The `div` tag is mapped to `pe:div` class.
- plain text is mapped to `pe:text`.
- `input type=button` is mapped to `pe:button`, etc.

Examples

There are many mcml examples where you browse in the paracraft package. Most of them are written in mcml v1, however the syntax should be the same for mcml v2. Click to search paracraft package for mcml

Following is just some quick examples:

```
<pe:mcml>
<script type="text/npl" refresh="false">
<![CDATA[
globalVar = "test global var";
function OnButtonClick()
    _guihelper.MessageBox("refresh page now?", function()
        Page:SetValue("myBtn", "refreshed");
        Page:Refresh(0);
    end);
end

repeat_data = { {a=1}, {a=2} };
function GetDS()
    return repeat_data;
end
]]>
</script>
    <div align="right" style="background-color:#ff0000;margin:10px;padding:10px;min-
↪width:400px;min-height:150px">
        <div align="right" style="padding:5px;background:url(Texture/Aries/Common/
↪ThemeKid/btn_thick_hl_32bits.png:7 7 7 7)">
            <div style="float:left;background-color:#000000;color:#ffd800;font-
↪size:20px;">
                hello world<font color="#ff0000">fontColor</font>
            </div>
            <div style="float:left;background-color:#0000ff;margin-left:10px;
↪width:20px;height:20px;">
            </div>
        </div>
        <div valign="bottom">
            <div style="float:left;background-color:#00ff00;width:60px;height:20px;">
            </div>
            <div style="float:left;background-color:#0000ff;margin-left:10px;
↪width:20px;height:20px;">
            </div>
        </div>
    </div>
    <div>
        hello world<font color="#ff0000">fontColor</font>
        <span color="#ff0000">fontColor</span>
    </div>
    <span color="#ff0000">
        <%=Eval('globalVar')%>
        <%=document.write('hello world')%>
    </span>
    <pe:container style="padding:5px;background-color:#ff0000">
        button:<input type="button" name="myBtn" value="RefreshPage" onclick=
↪"OnButtonClick" style=""/>
        editbox:<input type="text" name="myEditbox" value="" style="margin-left:2px;
↪width:64px;height:25px"/><br/>
        pe_if: <pe:if condition='<%=globalVar=="test global var"%>'>true</pe:if>
    </pe:container>
    <pe:repeat value="item in repeat_data" style="float:left">
        <div style="float:left;"><%=item.a%></div>
    </pe:repeat>
    <pe:repeat value="item in GetDS()" style="float:left">
        <div style="float:left;"><%=item.a%></div>
    </pe:repeat>
    <pe:repeat DataSource='<%=GetDS()%>' style="float:left">
        <div style="float:left;"><%=a%></div>
    </pe:repeat>
</pe:mcml>
```

MCML Layout

Please see [[System.Window]] first for basic alignment types. There are over 12 different alignment types, which supports automatic resizing of controls according to parent window size.

- mcml v1 have a `<pe:container>` control which supports all alignment types via `alignment` attribute.
- mcml v1 and v2's `<div>` supports a limited set of alignment type via `align` and `valign` attribute, where you can align object to left or right. Please note, due to mcml v1's one pass rendering algorithm, `right` alignment only works when parent width is fixed sized. mcml v2 does not have this limitation because it uses two pass algorithm. A workaround for mcml v1 is to use `<pe:container>` instead of `<div>` and use `alignment` for right alignment.

Relative positioning

Relative position is a way to position controls without occupying any space. It can be specified with `style="position:relative;"`. If all controls in a container is relatively positioned, they can all be accurately positioned at desired location relative to their parent.

However, relative position is NOT recommended, because a single change in child control size may result in changing all other child controls' position values and if parent control's size changes, the entire UI may be wrong. The recommended way of position is "float:left" or line based position. `<div>` defaults to line based position, in which it will automatically add `line break` at end. Specify "float:left" to make it float. Many other controls default to "float:left", such as `<input>`, `<span>`. Some UI designer in special product may still prefer relative position despite of this.

MCML vs HTML

MCML only supports a very limited subset of HTML. The most useful layout method in mcml is `<div>` tag.

NPL Code Behind and Embedding

In MCML page, there are three ways to embed NPL code.

- Use `<script src="helloworld.lua">`, this will load the npl file relative to the current page file.
- Embed NPL code section inside `<script>` anywhere in the page file

- Embed code in xml attribute in quotations like this `attrname='<%= %>'`. Please note it depends on whether the tag implementation support it. For each attribute you can not mix code with text. Unlike in NPL web server page, all code in MCML are evaluated at runtime rather than preprocessed. This is very important.

```
<script type="text/npl" src="somefile.lua" refresh="false">
<![CDATA[
globalVar = "test global var";
function OnButtonClick()
    _guihelper.MessageBox("refresh page now?", function()
        Page:SetValue("myBtn", "refreshed");
        Page:Refresh(0);
    end);
end
repeat_data = { {a=1}, {a=2} };
function GetDS()
    return repeat_data;
end
]]>
</script>
<div style='<%=format("width:%dpx", 100) %>'></div>
```

Page Variable Scoping

Global variables or functions defined in inline script are local to the page and not visible outside the page. So consider using a separate code behind file if one wants to expose values to external environment or simply your code is too long.

There is a predefined variable called `Page`, which is accessible to the page's sandbox's environment.

Two-Way Databinding

Many HTML developers have a `jquery` mindset, where they like to manipulate the DOM (Document Object Model or XML node) directly, which is **NOT** supported in mcml. Instead, one should have a `databinding mindset` like in `angularjs`. In MCML v1, `DOM != Controls`. DOM is only a static template for creating controls in a page. Controls can never modify static template unless you write explicit code, which is really rare.

Most mcml tags only support one-way data binding from static DOM template to controls only at the time when controls are created (i.e. Page Refresh, see below). To read value back from controls, one must either listen to `onchange` event of individual control or manually call `Page:GetValue(name)` for a collection of controls in a callback function, such as when user pressed `OK` button.

Two-way data-binding is useful but hard to implement in any system. Some HTML framework like `angularjs` does it by using a timer to periodically check watched data model and hook into every control's `onchange` event. One can simulate it manually in mcml. In mcml v2, DOM is almost equal to controls, hence it is relatively easy to implement two-way data binding in v2 than in v1.

> Automatic two-way data-binding is not natively supported in both v1 and v2. But we are planning to support it in the near future. The following example shows how to do manual two-way binding in mcml.

```
<pe:mcml>
<p>Two-way databinding Example in mcml:</p>
<script type="text/npl" refresh="false"><![CDATA[
-- nodes are generated according to this data source object, ds is global in page␣
↪scope so that it is shared between page refresh
ds = {
    block1 = nil,
    block2 = nil,
};

function asyncFillBlock1()
    PullData();
    -- use timer to simulate async API, such as fetching from a remote server.
    commonlib.TimerManager.SetTimeout(function()
        -- simulate some random data from remote server
        ds.block1 = {};
        for i=1, math.random(2,8) do
            ds.block1[i] = {key="firstdata"..i, value=false};
        end
        ds.block2 = nil;
        ApplyData();
    end, 1000)
end

function asyncFillBlock2()
    PullData();
    commonlib.TimerManager.SetTimeout(function()
        ds.block2 = {};

        -- count is based on the number of checked item in data1(block1)
        local data1_count = 0;
        for i, data in pairs(ds.block1) do
            data1_count = data1_count + (data.value and 1 or 0);
        end

        -- generate block2 according to current checked item in data1(block1)
        for i=1, data1_count do
            ds.block2[i] = {key="seconddata"..i, value=false};
        end
        ApplyData();
    end, 1000)
end

-- apply data source to dom
function ApplyData()
    Page("#block1"):ClearAllChildren();
    Page("#block2"):ClearAllChildren();

    if(ds.block1) then
        local parentNode = Page("#block1");
        for i, data in pairs(ds.block1) do
            local node = Page("<span />", {attr={style="margin:5px"}, data.key });
            parentNode:AddChild(node);
            local node = Page("<input />", {attr={type="checkbox", name=data.key,␣
↪checked = data.value and "true"} });
            parentNode:AddChild(node);
        end
        local node = Page("<input />", {attr={type="button", value="FillBlock2",␣
↪onclick = "asyncFillBlock2"} });
        parentNode:AddChild(node);
    end

    if(ds.block2) then
        local parentNode = Page("#block2");
        for i, data in pairs(ds.block2) do
```

```
            local node = Page("<span />", {attr={style="margin:5px"}, data.key });
            parentNode:AddChild(node);
            local node = Page("<input />",  {attr={type="text", EmptyText="enter text
↪", name=data.key, value=data.value, style="width:80px;"} });
            parentNode:AddChild(node);
        end
    end

    Page:Refresh(0.01);
end

-- pull data from DOM
function PullData()
    if(ds.block1) then
        for i, data in pairs(ds.block1) do
            data.value = Page:GetValue(data.key);
        end
    end
    if(ds.block2) then
        for i, data in pairs(ds.block2) do
            data.value = Page("#"..data.key):value(); -- just another way of␣
↪Page:GetValue()
        end
    end
end

function OnClickSubmit()
    -- TODO: one can also use a timer and hook to every onchange event of controls to␣
↪automatically invoke Apply and Pull Data.
    -- Here we just need to manually call it.
    PullData();
    _guihelper.MessageBox(ds);
end
]]></script>
<div>
    <input type="button" value="FillBlock1" onclick="asyncFillBlock1()"/><br />
    block1: <div name="block1"></div>
    block2: <div name="block2"></div>
</div>
<input type="button" value="Submit" onclick="OnClickSubmit()" /><br />
</pe:mcml>
```

In above example, When user clicks `FillBlock1` button, some random data is filled in `ds.block1`, each data is displayed as checkbox, and the user can check any number of them and click the `FillBlock2` button. This time we will fill `ds.block2` with same number of checked items in `ds.block1`, and each item is displayed as a text input box. When user clicks `Submit` button, the current `ds` is displayed in a message box with user filled data.

We call `ApplyData` to rebuild all related DOM whenever `ds` is modified externally, and we call `PullData` in the first line of each callback function so that our `ds` data is always up to date with mcml controls. The above example is called manual two-way binding because we have to call `ApplyData` and `PullData` manually. One can also use a timer and hook to every onchange event of controls to automatically invoke `ApplyData` and `PullData`, so that we do not have to write them everywhere in our callback functions.

> There is an old two-way data binding code called BindingContext, which is only used in `<form>` tag in mcml, it may cause some strange behaviors when you are manipulating DOM.

Page Refresh

For dynamic pages, it is very common to change some variable in NPL script and refresh the page to reflect the changes.

`<script refresh='false'>` means that the code section will not be reevaluated when page is refreshed. Also note that global variables or functions are shared between multiple script sections or multiple page refreshes.

Please also note that code changes in inline script blocks are usually reparsed when window is rebuild. However, if one use code behind model, npl file is cached and you need to restart your application to take effect. It is good practise to write in inline script first, and when code is stable move them to a code behind page. This will also make your page load faster during frequent page refresh or rebuild.

Writing Dynamic Pages

MCML has its own way of writing dynamic pages. In short, mcml uses frequent full `page refresh` to build responsive UI layout. Please note that each mcml page refresh does NOT rebuild (reparse) the entire DOM tree again, it simply reuses the old DOM tree and re-execute any embedded code on them each time. If the code is marked as `refresh="false"` (see last section), its last calculated value will be used.

Generally, MCML v1 uses a single pass to render the entire web page. MCML v2 uses two passes (click here to see details).

Unlike HTML/Javascript(especially jquery) where programmers are encouraged to manipulate the document tree directly, in mcml, you are NOT advised to change the DOM with your own code (although we do provide a jquery like interface to its DOM). Instead, we encourage you to data-bind your code and create different code paths using conditional component like `<pe:if>`, when everything is set you do a full page refresh with `Page:refresh()`. This is a little bit like `AngularJS`, but we do it with real-time executed inline code. The difference is that `AngularJS` will compile all tags and convert everything from angular DOM to HTML DOM, but mcml just does one pass rendering of the mcml tags and execute the code as it converts DOM to real NPL controls.

Another thing is that unlike in AngularJS where data changes are watched and page is automatically refreshed, in mcml, one has to manually call `Page:refresh(delayTimeSeconds)` to refresh the whole page, such as when user pressed a button to change the layout. Please note, changing the DOM does not automatically refresh the page, one still need to call the refresh function. Mcml code logics is much simpler to understand and learn, while angularJS, although powerful, but with too many hidden dark-magic that programmers will never figure out on their own.

Finally, mcml page is NOT preprocessed into a second DOM tree before rendering like what NPL web server page or PHP does. Every tag in mcml is a real mcml control or PageElement and the DOM tree is persistent across page refresh. MCML uses a single pass to render all of them. This also makes full page `Page:refresh()` much faster than other technologies. Please note, mcml v2 has even faster page refresh than mcml v1, this is because in mcml v2, NPL controls are pure script objects, while in v1 they are ParaUIObject in ParaEngine. The former is much faster to create/destroy or even shared during page refresh depending on each PageElement's implementation.

MCML Controls

Remember mcml is meant to be replacement for tedious control creation in the whole application, not a stateless and isolated web page. Each mcml page is an integral part of the application, they run in the same thread and have access to everything in that thread. They usually have direct interactions (or data-binding) with the 3D scene and other UI controls. To get the underlying control, one can use `Page:FindControl(name)`

MCML Page:Refresh(DelayTime)

`Page:Refresh(DelayTime)` refresh the entire page after DelayTime seconds. Please note that during the delay time period, if there is another call to this function with a longer delay time, the actual refresh page activation will be further delayed Note: This function is usually used with a design pattern when the MCML page contains asynchronous content such as pe:name, etc. whenever an asynchronous tag is created, it will first check if data for display is available at that moment. if yes, it will just display it as static content; if not, it will retrieve the data with a callback function. In the callback function, it calls this Refresh method of the associated page with a delay time. Hence, when the last delay time is reached, the page is rebuilt and the dynamic content will be accessible by then.

- @param DelayTime: if nil, it will default to self.DefaultRefreshDelayTime (usually 1.5 second).
- tip: If one set this to a negative value, it may causes an immediate page refresh.

Page Styling and Themes

MCML has limited support of inline css in `style` or `class` parameter. MCML v1 and v2 support style tag, in which we can define inline or file based css in the form of a standard NPL table. Please note, the syntax is not `text/css`, but `text/mcss`. It is simply a table object with key name, value pairs. The key name can be tag class name or class name, compound names with `tag, id, class` filtering is not supported.

Here is an example of defining styles with different class names, all mcml tags can have zero or more class names.

```
<pe:mcml>
<style type="text/mcss" src="script/ide/System/test/test_file_style.mcss">
{
    color_red = { color = "#ff0000" },
    color_blue = { color = "#0000ff", ["margin-left"] = 10 },
    bold = { ["font-weight"]="bold"    }
}
</style>
<span class="color_blue bold">css class</span>
</pe:mcml>
```

External css files are loaded and cached throughout the lifetime of the application process. Its content is also NPL table, such as `script/ide/System/test/test_file_style.mcss`:

```
-- Testing mcml css: script/ide/System/test/test_file_style.mcss
{
["default"] = { color="#ffffff" },
["mobile_button"] = {
    background = "Texture/Aries/Creator/Mobile/blocks_UI_32bits.png;1 1 34 34:12 12␣
↪12 12",
},
color_red = { color = "#ff0000" },
color_blue = { color = "#0000ff", ["margin-left"] = 10, ["font-size"] = 16 },
bold = { ["font-weight"] = "bold"    },
["pe:button"] = {padding = 10, color="#00ff00"},
}
```

In most cases, we can style the object directly in style attributes using 9 tiled image file. See next section.

To change the style of unthemed tags or buildin classes, one must progammatically modify the following table:

- In mcml v1, there is a global table like this one
- In mcml v2, there is a class called StyleDefault

9-Tile Image

MCML supports a feature called image tiling. It can use a specified section of an image and divide the subsection into 9 images and automatically stretch the 5 middle images as the target resizes (4 images at the corner are unstretched). This is very useful to create good looking buttons with various sizes.

The syntax is like this: `<div style="background:url(someimage_32bits.png#0 0 64 21:  10 10 10 10);" >`

- `#left top width height` is optional, which specifies a sub region. if ignored it means the entire image.
- `:left top to_right to_bottom` means distance to left, top, right and bottom of the above sub region.
- `_32bits.png` means that we should use 32bits color, otherwise we will compress the color to save video memory
- Image size must of order of 2, like 2,4,8,16,64, 128, 256, etc. Because `directX/openGL` only supports order of 2 images. If your image file is not order of 2, they will be stretched and become blurred when rendered.

Please note, in css file or `<style>` tag, # can also be `;` when specifying a sub region.

Setting Key Focus

For mcml v1, one can set key focus by following code. It only works for text box.

```
local ctl = Page:FindControl(name);
if(ctl) then
   ctl:Focus();
end
```

For input text box, one can also use.

```
<input type="text" autofocus="true"  ... />
```

But there can only be one such control in a window. If there are multiple ones, the last one get focus. For example

```
<pe:mcml>
    <div>
        Text: <input type="text" name="myAutoFocusEditBox" />
    </div>
    ... many other div here...
    <!-- This should be the last in the page -->
    <script type="text/npl" refresh="true">
        local ctl = Page("#myAutoFocusEditBox"):GetControl()
        ctl:Focus();
        ctl:SetCaretPosition(-1);
    </script>
</pe:mcml>
```

Conclusion

MCML v2 implementation is not very complete yet. You are welcome to contribute your own tags or controls to our main package.

FAQ

Why large font size appears to be blurred in mcml?

MCML uses 3D API to do all the graphical drawing. For each font size, an image must be loaded into video memory. Thus by default, mcml may scale an existing font to prevent creating unnecessary image. To prevent this behavior, one can use `style="base-font-size:30;font-size:30"` this will always create a base font image for your current font setting. In general, an application should limit the number of font settings used in your graphical application.

References

Documentation For MCML V1

- mcml v1 Source Code, such as:
  - pe_html
  - pe_editor
  - pe_html_input
  - pe_gridview: more examples
  - pe_treeview
  - pe_design
  - pe_component
  - pe_script
- Test cases(Examples) for mcml v1
  - test_pe_gridview

  More over, many existing GUI of paracraft are actually created with mcml v1. Search the source code for any HTML pages for examples.

CHAPTER 46

3D Programming

NPL/ParaEngine provides rich set of graphical API and libraries to create sophisticated interactive 3D application. ParaEngine used to be a full-featured computer game engine, it is now built-in with NPLRuntime. ParaEngine implementation is C/C++ based. All of its API can be accessed via NPL scripts.

Example 3D Projects

- Paracraft: is a 3D animation software written completely in NPL with over half million lines of NPL code.
- Haqi: is a 3D MMORPG written completely in NPL with over 1 million lines of NPL code.

CHAPTER 47

File Format

ParaEngine/NPL support several build-in file format. Some is used for 3d models, some for 3d world

3D Model file format

ParaEngine supports following built-in or standard file format.

- ParaX format (*.x): This is our build-in file format for advanced animated characters, particles, etc. One needs to download and install ParaEngine exporter plugin for 3dsmax 9 (32bits/64bits). However, we are not supporting the latest version of 3dsmax. Please consider using FBX file format.
- FBX format (`*.fbx`): FBX is the universal file format for AutoDesk product like MAYA/3dsmax. It is one of the most supported lossless file format in the industry. Click here for how to use FPX. (Please note model size and embedded texture matters)

  we only support FBX version 2013 or above. Version 2014, 2015 are tested.
- BMAX (*.bmax): This is our built-in block max file format, which is used exclusively by Paracraft application for storing static and animated blocks.
- STL format (*.stl): This is a file format used for 3d printing.

BMax file format

BMAX is short for Block Max, it is a file format used exclusively in Paracraft for storing and exchanging block data. In paracraft, the world is made up of blocks, each block may contain `x,y,z, block_id, block_data, custom_data`

- `x,y,z` is block position
- `block_id` is type id of the block, see block_types.xml for a complete list of block ids.
- `block_data` is a 32bits data of the block, it usually denotes the orientation or type of the block.

- `custom_data` can be any NPL table object that stores additional data of the block. Most static blocks does not contain custom_data, however, blocks like movie block, command blocks will save animation data and commands in this place.

You can select some blocks in paracraft and save them to bmax file to examine a real bmax file, it should be self-explanatory. Following is an example bmax file(containing a button, 2 movie blocks, a repeater and a wire):

```
<pe:blocktemplate>
    <pe:blocks>{
{-2,0,-3,105,5,},

{-2,0,-2,228,[6]={{name="cmd","/t 6 /end",},{{"{timeseries={lookat_z={times={0,5348,},
↪data={20001.56125,20004.65625,},ranges={{1,2,},},type=\"Linear\",name=\"lookat_z\",}
↪,eye_liftup={times={0,5348,},data={0.28568,0.26068,},ranges={{1,2,},},type=\"Linear\
↪",name=\"eye_liftup\",},lookat_x={times={0,5348,},data={20000.51959,20000.58203,},
↪ranges={{1,2,},},type=\"Linear\",name=\"lookat_x\",},eye_rot_y={times={0,5348,},
↪data={-3.11606,-3.11668,},ranges={{1,2,},},type=\"LinearAngle\",name=\"eye_rot_y\",}
↪,is_fps={times={0,5348,},data={0,0,},ranges={{1,2,},},type=\"Discrete\",name=\"is_
↪fps\",},lookat_y={times={0,5348,},data={-127.08333,-127.08333,},ranges={{1,2,},},
↪type=\"Linear\",name=\"lookat_y\",},eye_dist={times={0,5348,},data={8,8,},ranges={
↪{1,2,},},type=\"Linear\",name=\"eye_dist\",},has_collision={times={0,5348,},data={1,
↪1,},ranges={{1,2,},},type=\"Discrete\",name=\"has_collision\",},eye_roll={times={0,
↪5348,},data={0,0,},ranges={{1,2,},},type=\"LinearAngle\",name=\"eye_roll\",},},}",
↪name="slot",attr={count=1,id=10061,},},{"{timeseries={time={times={},data={},ranges=
↪{},type=\"Linear\",name=\"time\",},music={times={},data={},ranges={},type=\
↪"Discrete\",name=\"music\",},tip={times={},data={},ranges={},type=\"Discrete\",
↪name=\"tip\",},movieblock={times={},data={},ranges={},type=\"Discrete\",name=\
↪"movieblock\",},cmd={times={},data={},ranges={},type=\"Discrete\",name=\"cmd\",},
↪blocks={times={},data={},ranges={},type=\"Discrete\",name=\"blocks\",},text={times=
↪{},data={},ranges={},type=\"Discrete\",name=\"text\",},},}",name="slot",attr=
↪{count=1,id=10063,},},name="inventory",{"{timeseries={blockinhand={times={},data={},
↪ranges={},type=\"Discrete\",name=\"blockinhand\",},x={times={0,},data={20000.51959,}
↪,ranges={{1,1,},},type=\"Linear\",name=\"x\",},pitch={times={},data={},ranges={},
↪type=\"LinearAngle\",name=\"pitch\",},y={times={0,},data={-127.08333,},ranges={{1,1,
↪},},type=\"Linear\",name=\"y\",},parent={times={},data={},ranges={},type=\
↪"LinearTable\",name=\"parent\",},roll={times={},data={},ranges={},type=\
↪"LinearAngle\",name=\"roll\",},block={times={},data={},ranges={},type=\"Discrete\",
↪name=\"block\",},scaling={times={},data={},ranges={},type=\"Linear\",name=\"scaling\
↪",},gravity={times={},data={},ranges={},type=\"Discrete\",name=\"gravity\",},
↪HeadUpdownAngle={times={},data={},ranges={},type=\"Linear\",name=\"HeadUpdownAngle\
↪",},anim={times={},data={},ranges={},type=\"Discrete\",name=\"anim\",},bones={R_
↪Forearm_rot={times={0,34,1836,},data={{0.00024,0.00175,-0.01261,0.99992,},{0.00024,
↪0.00175,-0.01261,0.99992,},{0.00022,-0.05946,0.42959,0.90107,},},ranges={{1,3,},},
↪type=\"Discrete\",name=\"R_Forearm_rot\",},R_UpperArm_rot={times={0,34,1836,},data={
↪{-0.01933,-0.00286,0.03036,0.99935,},{-0.01802,-0.03834,0.32796,0.94375,},{-0.0137,-
↪0.01077,0.14324,0.98954,},},ranges={{1,3,},},type=\"Discrete\",name=\"R_UpperArm_
↪rot\",},isContainer=true,},speedscale={times={},data={},ranges={},type=\"Discrete\",
↪name=\"speedscale\",},assetfile={times={0,},data={\"actor\",},ranges={{1,1,},},
↪type=\"Discrete\",name=\"assetfile\",},skin={times={},data={},ranges={},type=\
↪"Discrete\",name=\"skin\",},z={times={0,},data={20001.56125,},ranges={{1,1,},},
↪type=\"Linear\",name=\"z\",},facing={times={},data={},ranges={},type=\"LinearAngle\
↪",name=\"facing\",},HeadTurningAngle={times={},data={},ranges={},type=\"Linear\",
↪name=\"HeadTurningAngle\",},name={times={0,},data={\"actor3\",},ranges={{1,1,},},
↪type=\"Discrete\",name=\"name\",},opacity={times={},data={},ranges={},type=\"Linear\
↪",name=\"opacity\",},},tooltip=\"actor3\",}",name="slot",attr={count=1,id=10062,},},
↪},name="entity",attr={bz=19201,bx=19200,class="EntityMovieClip",item_id=228,by=5,},}
↪,},

{-2,0,-1,197,3,},
{-2,0,0,189,},
{-2,0,1,228,[6]={{name="cmd","/t 30 /end",},{name="inventory",{name="slot",attr=
↪{count=1,id=10061,},},},name="entity",attr={bz=19204,bx=19200,class="EntityMovieClip
↪",item_id=228,by=5,},},},}

    </pe:blocks>
</pe:blocktemplate>
```

For user manual of using movie blocks, please see the courses in

- http://www.paracraft.cn/learn/movieblockcourses?lang=zh

ParaX File format

ParaX is binary file format used exclusively in NPL Runtime for animated 3D character asset.ParaX is like a concise version of FBX file (FBX is like BMAX file used by autodesk 3dsmax/maya, ...) NPL Runtime also support reading FBX file and load as ParaXModel object in C++, for user guide see FBX

- BMAX to ParaX converter in C++ (Lacking support for animation data in movie block) https://github.com/LiXizhi/NPLRuntime/tree/master/Client/trunk/ParaEngineClient/BMaxModel
- ParaX models:
  - https://github.com/LiXizhi/NPLRuntime/blob/master/Client/trunk/ParaEngineClient/ParaXModel/XFileCharModelParser.h
  - https://github.com/LiXizhi/NPLRuntime/blob/master/Client/trunk/ParaEngineClient/3dengine/ParaXSerializer.h
  - https://github.com/LiXizhi/NPLRuntime/blob/master/Client/trunk/ParaEngineClient/3dengine/ParaXSerializer.h

BMAX movie blocks to ParaX File format Conversion Details

- locate all movie blocks in BMAX file and ignore all other blocks. Movie blocks needs to be in the same order of 3D space (as repeater and wires indicates)
- The first movie block is always animation id 0 (idle animation)
- The second movie block is always animation id 4 (walk animation)
- The animation id of the third or following movie blocks can be specified in its movie block command
- The BMAX model(s) in the first movie block is used for defining bones and mesh (vertices, etc), please see this code for how to translate blocks into bones, vertices and sub meshes. BMAX models in all other movie blocks are ignored.
- Animation data in movie blocks are used to generate bone animations for each animation id in the final ParaX-Model.

3D World File Format

ParaEngine/NPL can automatically save or load 3D scene object to or from disk files.

To create an empty world, one can use

```
    local worldpath = "temp/clientworld";
    ParaIO.DeleteFile(worldpath.."/");
    ParaIO.CreateDirectory(worldpath.."/");
    ParaWorld.NewEmptyWorld(worldpath, 533.3333, 64);
```

3D world is tiled. Each tile is usually 512*512 meters. For historical reasons, it is 533.3333 by default. You will notice three files, after calling above function.

`temp/clientworld/worldconfig.txt` which is the entry file for 3d world, its content is like

```
-- Auto generated by ParaEngine
type = lattice
TileSize = 533.333313
(0,0) = temp/clientworld/flat.txt
(0,1) = temp/clientworld/flat.txt
...
```

It contains a mapping from tile (x,y) to tile configuration file.

`temp/clientworld/flat.txt` is configuration file for a single terrain tile.

```
-- auto gen by ParaEngine
Heightmapfile = temp/clientworld/flat.raw
MainTextureFile = terrain/data/MainTexture.dds
CommonTextureFile = terrain/data/CommonTexture.dds
Size = 533.333313
ElevScale = 1.0
Swapvertical = 1
HighResRadius = 30
DetailThreshold = 50.000000
MaxBlockSize = 64
DetailTextureMatrixSize = 64
NumOfDetailTextures = 0
```

Global Terrain Format

Global Terrain Format is a LOD triangle tile based terrain engine. The logical tile size is defined by 3d world configuration file, such as 533.333, however, its real dimension is always 512*512 and saved as a `*.raw` file.

Block World File Format

ParaEngine has a build-in block engine for rendering 3D blocky world. Block world is also tiled. The logical tile size is defined by 3d world configuration file, such as 533.333, however, its real dimension is always `512x512` and saved as a binary block region file in `*.raw` extension. Please note that block world region file `*.raw` is different from global terrain's `*.raw` file. The latter is just height maps.

Currently, each block may contain `x,y,z, block_id, block_data[, custom_data]`, currently `custom_data` is not supported in block region `*.raw` file. Block region file is encoded (compressed) using `increase-by-integer` algorithm, which will save lots of disk space if blocks are of the same type in the tiled region.

See BlockRegion.cpp for details

References

- http://www.paracraft.cn/ : see bmax model tutorial video
- https://github.com/LiXizhi/NPLRuntime/wiki
- https://github.com/LiXizhi/STLExporter
- http://wikicraft.cn/wiki/mod/packages : see STLExporter
- BMAX to ParaX converter in C++ (Lacking support for animation data in movie block) https://github.com/LiXizhi/NPLRuntime/tree/master/Client/trunk/ParaEngineClient/BMaxModel
- ParaX models:
  - https://github.com/LiXizhi/NPLRuntime/blob/master/Client/trunk/ParaEngineClient/ParaXModel/XFileCharModelParser.h
  - https://github.com/LiXizhi/NPLRuntime/blob/master/Client/trunk/ParaEngineClient/3dengine/ParaXSerializer.h

CHAPTER 48

3D Scene

There are a number of 3D objects which one can create via scripting API, such as mesh, physics mesh, sky box, camera, biped character, light, particles, containers, overlays, height map terrain, blocks, animations, shaders, 3D assets, etc.

Triangle Limitations

One can set the maximum number of total triangles of animated characters in the 3d scene per frame, like below

```
ParaScene.GetAttributeObject():SetField("MaxCharTriangles", 150000);
```

Faraway objects exceeding this value will be skipped during rendering. Another way of reducing triangle count is via level-of-detail (LOD) when loading mesh or characters. Currently, one must manually provide all LOD of mesh when loading a 3d file.

Rendering Pipeline

Currently all fixed function, shaders and deferred shading share the same predefined rendering pipelines. The render order can be partially affected by `RenderImportance`, `RenderOrder`, several `Shader files` property. But in general, it is a fixed general purpose rendering pipeline suitable in most situations.

The source code of the pipeline is hard-coded in `SceneObject.cpp's AdvanceScene()` function.

Here is what it does:

For each active viewport, we do the following:

- traverse the scene and quad-tree tile manager and call `PrepareRender()` for each visible scene object, which will insert the object into a number of predefined global render queues.
- `PIPELINE_3D_SCENE`
  - draw `MiniSceneGraph` with local cameras and render into textures

  - draw owner-draw objects like render targets.
  - draw all m_mirrorSurfaces into local textures
  - tesselate `global terrain` according to current camera
  - render shadow map for shadow casters queue, and other global object like blocks that cast shadows.
  - draw block engine multiframe world texture
  - render `global terrain`
  - draw opaque blocks in block engine
  - draw alpha-test enabled blocks in block engine
  - draw static meshes in big mesh queue from front to back
  - draw static meshes in small mesh queue from back to front
  - draw sprite objects queue
  - draw animated characters queue
  - render selection queue
  - render current sky object
  - draw block engine multiframe world texture on sky
  - draw missile object queue
  - draw block engine water reflection pass
  - block engine deferred shading post processing for opaque objects
  - draw batched transparent particles
  - block engine draw all alpha blended blocks
  - draw transparent animated characters in transparent biped queue
  - block engine deferred shading post processing for alpha blended object
  - draw block engine deferred lights
  - draw all head-on display
  - draw simulated global ocean surface
  - draw transparent static mesh objects in transparent mesh queue
  - draw transparent triangle face groups
  - draw objects in `post render queue`
  - draw particle systems
  - invoke custom post rendering shaders in NPL script for 3d scene
  - draw water waves
  - draw helpers for physics world debugging
  - draw helpers for bounding boxes of scene objects
  - draw portal system
  - draw overlays
- `PIPELINE_UI`

  - render all visible GUI objects by traversing the `GUIRoot` object.

- `PIPELINE_POST_UI_3D_SCENE`
  - only `MiniSceneGraph` whose `GetRenderPipelineOrder() == PIPELINE_POST_UI_3D_SCENE` will be rendered in this pipeline

- `PIPELINE_COLOR_PICKING`

Pipeline customization

- `RenderImportance` property of ParaObject only affects render order in a given queue.
- `RenderOrder` property of ParaObject will affect which queue the object goes into, as well as the order inside the queue. However, only `RenderOrder>100` is used to insert objects to `post render queue`.

For example, the following code will ensure the two objects are rendered last and ztest is disabled.

```
    local asset = ParaAsset.LoadStaticMesh("","model/common/editor/z.x")
    local obj = ParaScene.CreateMeshPhysicsObject("blueprint_center", asset, 1,1,1,␣
↪false, "1,0,0,0,1,0,0,0,1,0,0,0");
    obj:SetPosition(ParaScene.GetPlayer():GetPosition());
    obj:SetField("progress",1);
    obj:GetEffectParamBlock():SetBoolean("ztest", false);
    obj:SetField("RenderOrder", 101)
    ParaScene.Attach(obj);

    local player = ParaScene.CreateCharacter ("MyPlayer1", ParaAsset.LoadParaX("",
↪"character/v3/Elf/Female/ElfFemale.x"), "", true, 0.35, 0, 1.0);
    local x,y,z = ParaScene.GetPlayer():GetPosition()
    player:SetPosition(x+1,y,z);
    player:SetField("RenderOrder", 100)
    player:GetEffectParamBlock():SetBoolean("ztest", false);
    ParaScene.Attach(player);
```

CHAPTER 49

Camera

ParaEngine has build-in C++ support for a very versatile camera object, called AutoCamera. AutoCamera is the default camera used when you are inside a 3d world.

It can be configured in various ways to handle most key/mouse input for you, as well as controlling the biped object that it is focusing on.

To get or set current camera's attributes, one can use following code. Use Object Browser in NPL Code Wiki to see all of its attributes. Camera is a child of Scene object.

```
local attr = ParaCamera.GetAttributeObject()
```

Control Everything In Your Own Code

In most cases, we can focus the camera to a Biped player object in the scene to control the motion of the player directly. Trust me, writing a robust and smooth player camera controller is difficult, at least 3000 lines of code. If you really want to handle everything yourself, there is an option. Create a dummy invisible object to let the camera to focus to. And then set `IsControlledExternally` property to true for the dummy object.

```
your_dummy_obj:SetField("IsControlledExternally", true)
```

This way, the auto camera will not try to control the focused dummy object any more. And it is up to you to control the player, such as using `WASD` or arrow keys for movement and mouse to rotate its facing, etc.

Code Examples

Following are from `ParaEngineExtension.lua`. They are provided as an example.

```
-- set the look at position of the camera. It uses an invisible avatar as the camera␣
↪look at position.
-- after calling this function, please call ParaCamera.SetEyePos(facing, height,␣
↪angle) to change the camera eye position.
```

```
function ParaCamera.SetLookAtPos(x, y, z)
    local player = ParaCamera.GetDummyObject();
    player:SetPosition(x, y - 0.35, z);
    player:ToCharacter():SetFocus();
end

function ParaCamera.GetLookAtPos()
    return unpack(ParaCamera.GetAttributeObject():GetField("Lookat position", {0,0,0}
↪));
end
-- it returns polar coordinate system.
-- @return camobjDist, LifeupAngle, CameraRotY
function ParaCamera.GetEyePos()
    local att = ParaCamera.GetAttributeObject();
    return att:GetField("CameraObjectDistance", 0), att:GetField("CameraLiftupAngle",␣
↪0), att:GetField("CameraRotY", 0);
end

-- create/get the dummy camera object for the camera look position.
function ParaCamera.GetDummyObject()
    local player = ParaScene.GetObject("invisible camera");
    if(player:IsValid() == false) then
        player = ParaScene.CreateCharacter ("invisible camera", "", "", true, 0, 0,␣
↪0);
        --player:GetAttributeObject():SetField("SentientField", 0);--senses nobody
        player:GetAttributeObject():SetField("SentientField", 65535);--senses␣
↪everybody
        --player:SetAlwaysSentient(true);--senses everybody
        player:SetDensity(0); -- make it flow in the air
        player:SetPhysicsHeight(0);
        player:SetPhysicsRadius(0);
        player:SetField("SkipRender", true);
        player:SetField("SkipPicking", true);
        ParaScene.Attach(player);
        player:SetPosition(0, 0, 0);
    end
    return player;
end

-- set the camera eye position by camera object distance, life up angle and rotation␣
↪around the y axis. One must call ParaCamera.SetLookAtPos() before calling this␣
↪function.
-- e.g.ParaCamera.SetEyePos(5, 1.3, 0.4);
function ParaCamera.SetEyePos(camobjDist, LifeupAngle, CameraRotY)
    local att = ParaCamera.GetAttributeObject();
    att:SetField("CameraObjectDistance", camobjDist);
    att:SetField("CameraLiftupAngle", LifeupAngle);
    att:SetField("CameraRotY", CameraRotY);
end
```

CHAPTER 50

Block Engine

The block engines can manage multiple `BlockWorld` object.

BlockWorld

Each `BlockWorld` represents a block world with at most `32000x32000x256`, where 256 is height of the world. Each BlockWorld will dynamically and asynchronously load `BlockRegion` on demand.

BlockRegion

It manages 512x512x256 blocks, which are saved into a single file.

BlockChunk

It caches model and light Data for `16x16x16` region. Each chunk is converted and added to a queue into `BlockRenderTask` for sorting and rendering.

BlockLightGrid

It calculates sun and block lighting in a separate thread and save the result into `BlockChunk` for rendering.

BlockModel

BlockModel is usually cube 3D model, but it is not a 3D object directly used in rendering, instead it is actually used in `BlockTemplate` to provide rendering and physics data.

CHAPTER 51

NPL Web Server

Click here for step-by-step tutorial to build a NPL web site.

NPL Runtime provides a build-in self-contained framework for writing web server applications in a way similar to PHP, yet with asynchronous API like NodeJs.

Why Write Web Servers in NPL?

Turning Async Code Into Synchronous Ones

Many tasks on the server side has asynchronous API like database operations, remote procedure call, background tasks, timers, etc. Asynchronous API frees the processing thread from io-bound or long running task. Using asynchronous API (like those in `NodeJs`) makes web server fast, but difficult to write, as there are too many function callbacks that breaks the otherwise sequential and flat programming code.

In NPL page file, any asynchronous API can be used as synchronous ones via `resume/yield`, so that constructing a web page is like writing a document sequentially like in `PHP`. Under the hood, it is still asynchronous API and the same worker thread can process thousands of requests per second even some of them takes a long time to finish. Following is a quick example show the idea.

```
<?
local value = "Not Set"
-- any async function with callback
local mytimer = commonlib.Timer:new({callbackFunc = function(timer)
   value = "Set";
   resume();
end})
-- start the timer after 1000 milliseconds
mytimer:Change(1000, nil)

-- async wait when job is done
yield();
assert(value == "Set");
```

Please note, `resume/yield` is NOT the ones as in multithreaded programming where locks and signals are involved. It is a special `coroutine` internally; the code is completely lock-free and the thread is `NOT` waiting but processing other URL requests. Please see [[NPLServerPage]] for details.

Self-Contained Client/Server Solution

NPL Web Server is fast and very easy to deploy on all platforms (no external dependencies even for databases). You may have a very sophisticated 3d client application, which may also be servers and/or web servers, such as Paracraft.

We believe, every software either running on central server farm or endpoints like home or mobile devices should be able to provide services via TCP connection and/or HTTP (web server). In many of our applications, a single software acts both as client and server. Very few existing programming language provides an easy way or architecture for sharing client and server side code or variables in a lock-free fashion due to multi-threaded (multi-process) nature of those web frameworks. Moreover, a server application is usually hard to deploy and config, making it difficult to install on home computers or mobile devices.

NPL is a language to solve these problems and provides rich client side API as well as a web server framework that can service as many requests as professional web servers while sharing the entire runtime environment for your client and server code. See below.

Programming Model of NPL web server

NPL Runtime provides a build-in framework for writing web server applications in a way similar to PHP.

> NPL Web Server recommends the async programming model like Nodejs, but without losing the simplicity of synchronous mixed code programming like in PHP.

Following is an example `helloworld.page` server page.

| |query database and wait for database result | MVC Render | |—-|———————————————————-|——–—–| | duration |95% | 5% |

The processing of a web page usually consists of two phases.

- One is fetching data from database engine, which usually takes over 95% of the total time.
- The other is page rendering, which is CPU-bound and takes only 5% of total request time.

With NPL's `yield` method, it allows other web requests to be processed concurrently in the 90% interval while waiting database result on the `same` system-level thread. See following code to see how easy to mix async-code with template-based page rendering code. This allows us to serve `5000 requests/sec` in a single NPL thread concurrently, even if each request takes 30ms seconds to fetch from database.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
↪xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>NPL server page test. </title>
</head>
<body>
<p>
    <?npl
        echo "Hi, I'm a NPL script!";
    ?>
</p>

<?
```

```
-- connect to TableDatabase (a NoSQL db engine written in NPL)
db = TableDatabase:new():connect("database/npl/", function() end);

-- insert 5 records to database asynchronously.
local finishedCount = 0;
for i=1, 5 do
    db.TestUser:insertOne({name=("user"..i), password="1"}, function(err, data)
        finishedCount = finishedCount + 1;
        if(finishedCount == 5) then
            resume();
        end
    end);
end
yield(); -- async wait when job is done

-- fetch all users from database asynchronously.
db.TestUser:find({}, function(err, users)  resume(err, users); end);
err, users = yield(); -- async wait when job is done
?>

<?npl for i, user in ipairs(users) do ?>
    i = <?=i?>, name=<? echo(user.name) ?> <br/>
<?npl end ?>

<p>
    1.  <?npl echo 'if you want to serve NPL code in XHTML or XML documents, use␣
↪these tags'; ?>
</p>
<p>
    2.  <? echo 'this code is within short tags'; ?>
    Code within these tags <?= 'some text' ?> is a shortcut for this code <? echo
↪'some text' ?>
</p>
<p>
    3.  <% echo 'You may optionally use ASP-style tags'; %>
</p>

<% nplinfo(); %>

<% print("<p>filename: %s, dirname: %s</p>", __FILE__, dirname(__FILE__)); %>

<%
some_global_var = "this is global variable";

-- include file in same directory
include("test_include.page");
-- include file relative to web root directory. however this will not print, because␣
↪we use include_once, and the file is already included before.
include_once("/test_include.page");
-- include file using disk file path
local result = include(dirname(__FILE__).."test_include.page");

-- we can also get the result from included file
echo(result);
%>

</body>
</html>
```

```
<%
--assert(false, "uncomment to test runtime error");

function test_exit()
    exit();
end
test_exit();
assert(false, "can not reach here");
%>
```

Content of `test_include.page` referenced in above server page is:

```
<?npl

echo("<p>this is from included file: "..(some_global_var or "").."</p>");

-- can also has return value.
return "<p>from test_include.page</p>";
```

The output of above server page, when viewed in a web browser, such as `http://localhost:8099/helloworld` is

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
↪xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>NPL server page test. </title>
</head>
<body>
<p>
    Hi, I'm a NPL script!</p>

    i = 1, name=user1 <br/>
    i = 2, name=user2 <br/>
    i = 3, name=user3 <br/>
    i = 4, name=user4 <br/>
    i = 5, name=user5 <br/>
<p>
    1.  if you want to serve NPL code in XHTML or XML documents, use these tags</p>
<p>
    2.  this code is within short tags    Code within these tags some text is a␣
↪shortcut for this code some text</p>
<p>
    3.  You may optionally use ASP-style tags</p>

<p>NPL web server v1.0</p><p>site url: http://localhost:8099/</p><p>your ip: 127.0.0.1
↪</p><p>{
<br/>["Host"]="localhost:8099",
<br/>["rcode"]=0,
<br/>["Connection"]="keep-alive",
<br/>["Accept"]="text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;
↪q=0.8",
<br/>["Accept-Encoding"]="gzip, deflate, sdch",
<br/>["method"]="GET",
<br/>["body"]="",
<br/>["tid"]="~1",
<br/>["url"]="/helloworld.page",
<br/>["User-Agent"]="Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,␣
↪like Gecko) Chrome/48.0.2564.116 Safari/537.36",
<br/>["Upgrade-Insecure-Requests"]="1",
<br/>["Accept-Language"]="en-US,en;q=0.8",
<br/>}
<br/></p><p>filename: script/apps/WebServer/test/helloworld.page, dirname: script/
↪apps/WebServer/test/</p><p>this is from included file: this is global variable</p>
↪<p>this is from included file: this is global variable</p><p>from test_include.page
↪</p></body>
</html>
```

Web Server Source Code

- script/apps/WebServer: implementation of a web server (like Apache) in NPL.
- script/apps/WebServer/WebServer.lua: entry file
- script/apps/WebServer/admin: A php-like web site framework in NPL. See [[AdminSiteFramework]] for details

Introduction

The NPL web server implementation uses NPL's own messaging system, and is written in pure NPL scripts without any external dependencies. It allows you to create web site built with NPL Runtime on the back-end and `JavaScript/HTML5` on the client.

`NPL language service` plugins for visual studio (community edition is free) provides a good coding environment. See below:

Website Examples

- WebServerExample: a sample web site with NPL code wiki
- Wikicraft: full website example
- `script/apps/WebServer/admin` is a NPL based web site framework similar to the famous `WordPress.org`. It is recommended that you xcopy all files in it to your own web root directory and work from there. See [[AdminSiteFramework]] for details.
- `script/apps/WebServer/test` is a test site, where you can see the test code.

starting a web server programmatically

Starting server from a web root directory:

```
    NPL.load("(gl)script/apps/WebServer/WebServer.lua");
    WebServer:Start("script/apps/WebServer/test", "0.0.0.0", 8099);
```

Open `http://localhost:8099/helloworld.lua` in your web browser to test. See `log.txt` for details.

There can be a `webserver.config.xml` file in the web root directory. see below. if no config file is found, `default.webserver.config.xml` is used, which will redirect all request without extension to `index.page` and serve *.lua, *.page, and all static files in the root directory and it will also host NPL code wiki at [[http://127.0.0.1:8099]].

starting a web server from command line

You can bootstrap a webserver using the buildin file, run following commmand:

```
npl bootstrapper="script/apps/WebServer/WebServer.lua"  port="8099"
```

- config: can be omitted which defaults to `config/WebServer.config.xml` in current directory

Web Server configuration file

More info, see `script/apps/WebServer/test/webserver.config.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<!-- web server configuration file: this node can be child node, thus embedded in␣
↪shared xml -->
<WebServer>
  <!--which HTTP ip and port this server listens to. -->
  <servers>
    <!-- @param host, port: which ip port to listen to. if * it means all. -->
    <server host="*" port="8099" host_state_name="">
      <defaultHost rules_id="simple_rule"></defaultHost>
      <virtualhosts>
        <!-- force "http://127.0.0.1/" to match to iternal npl_code_wiki site for␣
↪debugging  -->
        <host name="127.0.0.1:8099" rules_id="npl_code_wiki" allow='{"127.0.0.1"}'></
↪host>
      </virtualhosts>
    </server>
  </servers>
  <!--rules used when starting a web server. Multiple rules with different id can be␣
↪defined. -->
  <rules id="simple_rule">
    <!--URI remapping example-->
    <rule match='{"^[^%.]+$", "robots.txt"}' with="WebServer.redirecthandler" params='
↪{"/index.page"}'></rule>
    <!--npl script example-->
    <!--<rule match="%.lua$" with="WebServer.makeGenericHandler" params='{docroot=
↪"script/apps/WebServer/test", params={}, extra_vars=nil}'></rule>-->
    <rule match='{"%.lua$", "%.npl$"}' with="WebServer.npl_script_handler" params='%CD
↪%'></rule>
    <!--npl server page example-->
    <rule match="%.page$" with="WebServer.npl_page_handler" params='%CD%'></rule>
    <!--filehandler example, base dir is where the root file directory is. %CD% means␣
↪current file's directory-->
    <rule match="." with="WebServer.filehandler" params='{baseDir = "%CD%"}'></rule>
  </rules>
</WebServer>
```

Each server can have a default host and multiple virtual hosts. Each host must be associated with a rule_id. The rule_id is looked up in the rules node, which contains multiple rules specifying how request url is mapped to their handlers. `npl_code_wiki` is an internal rule_id which is usually used for host `http://127.0.0.1:8099/` for debugging purposes only, see above example code. If you changed your port number, you need to update the config file accordingly, otherwise `npl_code_wiki` will not be available.

Each rule contains three attributes: match, with and params.

- "match" is a regular expresion string or a array table of reg strings like shown in above sample code.
- "with" is a handler function which will be called when url matches "match". More precisely, it is handler function maker, that returns the real handler function(request, response) end. When rules are compiled at startup time, these maker functions are called just once to generate the actual handler function.
- "params" is an optional string or table that will be passed to the handler maker function to generate the real handler function.

The rules are applied in sequential order as they appeared in the config file. So they are usually defined in the order of redirectors, dynamic scripts, file handlers.

Handler Makers

Some common handlers are implemented in `common_handlers.lua`. Some of the most important ones are listed below

WebServer.redirecthandler

It is the url redirect handler. it generate a handler that replaces "match" with "params"

```
    <rule match="^[^%./]*/$" with="WebServer.redirecthandler" params='{"readme.txt"}'>
↪</rule>
```

The above will redirect `http://localhost:8080/` to `http://localhost:8080/readme.txt`.

WebServer.filehandler

It generate a handler that serves files in the directory specified in "params" or "params.baseDir".

```
    <rule match="." with="WebServer.filehandler" params='{baseDir = "script/apps/
↪WebServer"}'></rule>
    alternatively:
    <rule match="." with="WebServer.filehandler" params='script/apps/WebServer'></
↪rule>
```

The above will map `http://localhost:8080/readme.txt` to the disk file `script/apps/WebServer/readme.txt`

WebServer.npl_script_handler

Currently it is the same as WebServer.makeGenericHandler. In future we will support remote handlers that runs in another thread asynchrounously. So it is recommended for user to use this function instead of Web-

Server.makeGenericHandler serving request using npl files to generate dynamic response. This is very useful for REST-like web service in XML or json format.

```
    <rule match="%.lua$" with="WebServer.npl_script_handler" params='script/apps/
↪WebServer/test'></rule>
    alternatively:
    <rule match="%.lua$" with="WebServer.npl_script_handler" params='{docroot="script/
↪apps/WebServer/test"}'></rule>
```

The above will map `http://localhost:8080/helloworld.lua` to the script file `script/apps/WebServer/test/helloworld.lua`

Caching on NPL Web Server

1. We use `File Monitor API` to monitor all file changes in web root directory.
2. For dynamic page files: recompile dynamic page if changed. All compiled *.page code is always in cache. So multiple requests calling the same page file only compile once.
3. For static files: everything is cached in either original or gzip-format ready for direct HTTP response. All static files are served using a separate NPL thread to prevent IO affecting the main thread.

   for more advanced file caching, one may consider using `nginx` as the gateway load balancer.

Caching in Application Code

For caching in local thread, there is a helper class called mem_cache.

```
NPL.load("(gl)script/apps/WebServer/mem_cache.lua");
local mem_cache = commonlib.gettable("WebServer.mem_cache");
local obj_cache = mem_cache:GetInstance();
obj_cache:add("name", "value")
obj_cache:replace("name", "value1")
assert(obj_cache:get("name") == "value1");
assert(obj_cache:get("name", "group1") == nil);
obj_cache:add("name", "value", "group1")
assert(obj_cache:get("name", "group1") == "value");
```

Caching in different computer

TODO: `mem_cache` can be configured to read/write data from a remote computer.

CHAPTER 52

NPL Server Page

NPL server page is a mixed HTML/NPL file, usually with the extension `.page`.

How Does It Work?

At runtime time, server page is preprocessed into pure NPL script and then executed. For example

```
<html><body>
<?npl  for i=1,5 do ?>
   <p>hello</p>
<?npl  end ?>
</body></html>
```

Above server page will be pre-processed into following NPL page script, and cached for subsequent requests.

```
echo ("<html><body>");
for i=1,5 do
 echo("<p>hello</p>")
end
echo ("</body></html>");
```

When running above page script, `echo` command will generate the final HTML response text to be sent back to client.

Sandbox Environment

When a HTTP request come and redirected to NPL page handler, a special sandbox environment table is created, all page scripts related to that request is executed in this newly created sandbox environment. So you can safely create global variables and expect them to be uninitialized for each page request.

However, the sandbox environment also have read/write access to the global per-thread NPL runtime environment, where all NPL classes are loaded.

`NPL.load` vs `include`

In a page file, one can call `NPL.load` to load a given NPL class, such as `mysql.lua`; or one can also use the page command `include` to load another page file into sandbox environment. The difference is that classes loaded by `NPL.load` will be loaded only once per thread; where `include` will be loaded for every HTTP request handled by its worker thread. Moreover, NPL web server will monitor file changes for all page files and recompile them when modified by a developer; for files with `NPL.load`, you need to restart your server, or use special code to reload it.

Mixing Async Code with Yield/Resume

The processing of a web page usually consists of two phases.

- One is fetching data from database engine, which usually takes over 95% of the total time.
- The other is page rendering, which is CPU-bound and takes only 5% of total request time.

| |query database and wait for database result | MVC Render | |—-|——————————————————————-|————| | duration |95% | 5% |

With NPL's `yield` method, it allows other web requests to be processed concurrently in the 90% interval while waiting database result on the `same` system-level thread. See following code to see how easy to mix async-code with template-based page rendering code. This allows us to serve `5000 requests/sec` in a single NPL thread concurrently, even if each request takes 30ms seconds to fetch from database.

Following is excerpt from our `helloworld.page` example.

```
<?
-- connect to TableDatabase (a NoSQL db engine written in NPL)
db = TableDatabase:new():connect("database/npl/", function() end);

-- insert 5 records to database asynchronously.
local finishedCount = 0;
for i=1, 5 do
    db.TestUser:insertOne({name=("user"..i), password="1"}, function(err, data)
        finishedCount = finishedCount + 1;
        if(finishedCount == 5) then
            resume();
        end
    end);
end
yield(); -- async wait when job is done

-- fetch all users from database asynchronously.
db.TestUser:find({}, function(err, users)  resume(err, users); end);
err, users = yield(true); -- async wait when job is done
?>

<?npl for i, user in ipairs(users) do ?>
    i = <?=i?>, name=<? echo(user.name) ?> <br/>
<?npl end ?>
```

Code Explanation: When the first `yield()` is called, the execution of page rendering is paused. It will be resumed by the result of a previous async task. In our case, when all five users have been inserted to our database, we will call `resume()`, which will immediately resume page execution from last yield (paused) code position.

Then we started another async task to fetch all users in the database, and called yield immediately to wait for its results. This time we passed some parameter `resume(err, users)`, everything passed to resume will be returned from yield() function. So `err, users = yield(true)` will return the `users` table when it returns.

Please note, we recommend you pass a boolean `err` as first parameter to `resume`, since all of our async API follows the same rule. Also note that we passed true to the second `yield` function, which tells to page renderer to output error and stop execution immediately. If you want to handle the error yourself, please pass nothing to yield like `err, users = yield()`

Page Commands

The following commands can only be called from inside an NPL page file. They are shortcut to long commands.

```
Following objects and functions can be used inside page script:
    request:   current request object: headers and cookies
    response:   current response object: send headers or set cookies, etc.
    echo(text):   output html
    __FILE__: current filename
    page: the current page (parser) object
    _GLOBAL: the _G itself

following are exposed via meta class:
    include(filename, bReload):  inplace include another script
    include_once(filename):  include only once, mostly for defining functions
    print(...):  output html with formatted string.
    nplinfo():   output npl information.
    exit(text), die():   end the request
    dirname(__FILE__):   get directory name
    site_config(): get the web site configuration table
    site_url(path, scheme):
    addheader(name, value):
    file_exists(filename):
    log(obj)
    sanitize(text)  escape xml '<' '>'
    json_encode(value, bUseEmptyArray)   to json string
    json_decode(str)  decode from json string
    xml_encode(value)    to xml string
    include_pagecode(code, filename):  inplace include page code.
    get_file_text(filename)
    util.GetUrl(url, function(err, msg, data) end):
    util.parse_str(query_string):
    err, msg = yield(bExitOnError)  pause execution until resume() is called.
    resume(err, msg)  in async callback, call this function to resume execution from␣
↪last yield() position.
```

see script/apps/WebServer/npl_page_env.lua for detailed documentation.

Request/Response Object

- request object: current request object: headers and cookies
- response object: current response object: send headers or set cookies, etc.

Memory Cache and Global Objects

NPL web server has a built-in simple local memory cache utility. One can use it to store objects that is shared by all requests on the main thread. In future it may be configured to run on a separate server like `memcached`.

See below:

```
NPL.load("(gl)script/apps/WebServer/mem_cache.lua");
local mem_cache = commonlib.gettable("WebServer.mem_cache");
local obj_cache = mem_cache:GetInstance();
obj_cache:add("name", "value")
obj_cache:replace("name", "value1")
assert(obj_cache:get("name") == "value1");
assert(obj_cache:get("name", "group1") == nil);
obj_cache:add("name", "value", "group1")
assert(obj_cache:get("name", "group1") == "value");
```

Alternatively, one can also create global objects that is shared by all requests in the NPL thread, by using `commonlib.gettable()` method. Table objects created with `commonlib.gettable()` is different from `gettable()` in page file. The latter will create table on the local page scope which lasts only during the lifetime of a given http request.

File Uploader

NPL web server supports `multipart/form-data` by which one can upload binary file to the server. It is recommended to use a separate server for file upload, because it is IO bound and consumes bandwidth when file is very large.

Click here for a complete example of file uploader.

Here is the client side code to upload file as `enctype="multipart/form-data"`

```
<form name="uploader" enctype="multipart/form-data" class="form-horizontal" method=
↪"post" action="/ajax/fileuploader?action=upload">
  <input name="fileToUpload" id="fileToUpload" type="file" class="form-control"/>
  <input name="submit" type="submit" class="btn btn-primary" value="Upload"/>
</form>
```

Here is an example NPL server page code, the binary contents of the file is in `request:getparams()["fileToUpload"].contents`. The maximum file upload size allowed can be configured by NPL runtime attribute. The default value is 100MB.

```
local fileToUpload = request:getparams()["fileToUpload"]
if(fileToUpload and request:getparams()["submit"] and fileToUpload.name and␣
↪fileToUpload.contents) then
    local target_dir = "temp/uploads/" .. ParaGlobal.GetDateFormat("yyyyMMdd") .. "/";
    local target_file = target_dir .. fileToUpload.name;
    local fileType = fileToUpload.name:match("%.(%w+)$"); -- file extension

    -- check if file already exists
    if(file_exists(target_file)) then
        response:send({err = "Sorry, file already exists."});
        return
    end
    -- check file size
    if (fileToUpload.size and fileToUpload.size> 5000000) then
```

```
        response:send({err = "Sorry, your file is too large."});
        return
    end

    -- Allow certain file formats
    if(false and fileType ~= "jpg" and fileType ~= "png" and fileType~="txt" ) then
        response:send({err = "Sorry, only JPG, PNG & TXT files are allowed."});
        return
    end

    -- if everything is ok, try to save file to target directory
    ParaIO.CreateDirectory(target_dir);
    local file = ParaIO.open(commonlib.Encoding.Utf8ToDefault(target_file), "w");
    if(file:IsValid()) then
        file:write(fileToUpload.contents, #fileToUpload.contents);
        file:close();
        response:send({contents = target_file, name = fileToUpload.name, size =␣
↪fileToUpload.size, ["content-type"] = fileToUpload["content-type"], });
        return;
    else
        response:send({err = "can not create file on disk. file name invalid or disk␣
↪is full."});
    end
end
response:send({err = "unknown err"});
```

Server Side Redirection

See also RFC standard

```
<?npl
response:set_header("Location", "http://www.paracraft.cn/")
response:status(301):send_headers();
```

CHAPTER 53

NPL Admin Site Framework

`WebServer/admin` is a open source NPL-based web site framework. It comes with the `main` package and contains all source code of [[NPLCodeWiki]]. It is served as a demo and debugger for your own web application. It is also by default run side by side on `127.0.0.1:8099/` with your website on `localhost:8099`. See default.webserver.config.xml.

How to run NPL Admin Site

```
npl script/apps/WebServer/WebServer.lua
```

or you can run with more options like

```
npl bootstrapper="script/apps/WebServer/WebServer.lua"  port="8099"
```

Once started, you can visit the admin site from [[http://localhost:8099]]

Introduction

I have implemented it according to the famous `WordPress.org` blog site. The default theme template is based on "sensitive" which is a free theme of wordpress. "sensitive" theme has no dependency on media and can adjust display and layout according to screen width, which is suitable for showing on mobile phone device.

How to use

Usage 1: Use by reference (Recommended)

Basically, if you want to use admin framework without cloning the file, simply add following line to your rule file

```
    <!--wp framework related js, css, files-->
    <rule match="^/?wp%-" with="WebServer.filehandler" params='{baseDir = "script/
↪apps/WebServer/admin/"}'></rule>
```

In your page `index.page` file, you can include the main file of the wp framework, such as this:

```
<?npl

-- we will not load complete framework, but only ajax and helper functions
WP_USE_MINI_LOADER = true;
include_once("script/apps/WebServer/admin/wp-main.page");

-- call your router, such as
include_once("./myproj/routes.page");
```

Usage 2: Use by cloning

- copy everything in this folder to your own web site root, say /www/MySite/
- modify wp-content/database/*.xml for site description and menus.
- add your own web pages to wp-content/pages/, which are accessed by their filename in the url.
- If you want more customization to the look, modify the wp-content/themes/sensitive or create your own theme folder. Remember to set your theme in `wp-content/database/table_sitemeta.xml`, which contains all options for the site.

Usage 3: Adding new features to it

- Simply put your own files in `script/apps/WebServer/admin/` in your dev or working directory, your files will be loaded first before the one in `npl_package/main`.

Architecture

```
The architecture is based on Wordpress.org (4.0.1). Although everything is rewritten␣
↪in NPL, I have kept all functions, filters, and file names identical to wordpress.
See `wp-includes/` for the framework source code.
```

Code locations: * framework loader is in `wp-settings.page` * site options: `wp-content/database/table_sitemeta.xml`: such as theme, default menu, etc. * menus: `wp-content/database/table_nav_menu.xml`

Ajax framework

```
Any request url begins with `ajax/xxx` use the `wp-admin/admin-ajax.page`. it will␣
↪automatically load the page xxx. and invoke `do_action('wp_ajax_xxx')`.
If the request url begins with `ajax/xxx?action=yyy`, then page xxx is loaded, and␣
↪`do_action('wp_ajax_xxx')` is invoked.
A page that handles ajax request needs to call `add_action('wp_ajax_xxx' function_
↪name)` to register a handler for ajax actions.
see `wp-content/pages/aboutus.page` for an example.
```

CHAPTER 54

Table Database

`Table database` is implemented in NPL and is the default and recommended database engine. It can be used without any external dependency or configuration.

Introduction to Table Database

A schema-less, server-less, NoSQL database able to process big data in multiple NPL threads, with extremly intuitive table-like API, automatic indexing and extremely fast searching, keeping data and algorithm in one system *just like how the human brain works*.

Local Storage API vs Full-Featured Database Solution

The raw table database has no configuration file. Please regard the raw table database as a local storage API provided by NPL, rather than a full-featured database solution. However its performance is good enough to be used as the database engine for medium sized projects. You need to write application-level code to have something like authentication, and data replication/hashing on multiple computers, etc. However, in future, some of these functions may be provided as additional libraries on top of the raw table database to provide enterprise level database solution that runs on multiple nodes.

Performance

Following is tested on my Intel-i7-3GHZ CPU. See test folder for test source code.

Run With Conservative Mode

Following is averaged value from 100000+ operations in a single thread

- Random non-index insert: `43478 inserts/second`

  - Async API tested with default configuration with 1 million records on a single thread.
- Round trip latency call:
  - Blocking API: `20000 query/s`
  - Non-blocking API: `11ms` or `85 query/s` (due to NPL time slice)
  - i.e. Round strip means start next operation after previous one is returned. This is latency test.
- Random indexed inserts: `17953 query/s`
  - i.e. start next operation immediately, without waiting for the first one to return.
- Random select with auto-index: `18761 query/s`
  - i.e. same as above, but with findOne operation.
- Randomly mixing CRUD operations: `965-7518` query/s
  - i.e. same as above, but randomly calling Create/Read/Update/Delete (CRUD) on the same auto-indexed table.
  - Mixing read/write can be slow when database grows bigger. e.g. you can get `18000 CRUD/s` for just 10000 records.

Run With Aggressive Mode

One can also use in-memory journal file or ignore OS disk write feedback to further increase DB throughput by 30-100% percent. See `Store.lua` for details. By default, this feature is off.

Code Examples:

```
    NPL.load("(gl)script/ide/System/Database/TableDatabase.lua");
    local TableDatabase = commonlib.gettable("System.Database.TableDatabase");
    -- this will start both db client and db server if not.
    local db = TableDatabase:new():connect("temp/mydatabase/", function() end);

    -- Note: `db.User` will automatically create the `User` collection table if not.
    -- clear all data
    db.User:makeEmpty({}, function(err, count) echo("deleted"..(count or 0)) end);
    -- insert 1
    db.User:insertOne(nil, {name="1", email="1@1",}, function(err, data)  assert(data.
↪email=="1@1")     end)
    -- insert 1 with duplicate name
    db.User:insertOne(nil, {name="1", email="1@1.dup",}, function(err, data) ␣
↪assert(data.email=="1@1.dup")     end)

    -- find or findOne will automatically create index on `name` and `email` field.
    -- indices are NOT forced to be unique. The caller needs to ensure this see␣
↪`insertOne` below.
    db.User:find({name="1",}, function(err, rows) assert(#rows==2); end);
    db.User:find({name="1", email="1@1"}, function(err, rows) assert(rows[1].email==
↪"1@1"); end);
    -- find with compound index of name and email
    db.User:find({ ["+name+email"] = {"1", "1@1"} }, function(err, rows) assert(
↪#rows==1); end);

    -- force insert
    db.User:insertOne(nil, {name="LXZ", password="123"}, function(err, data) ␣
↪assert(data.password=="123")  end)
```

```
    -- this is an update or insert command, if the query has result, it will actually␣
↪update first matching row rather than inserting one.
    -- this is usually a good way to force uniqueness on key or compound keys,
    db.User:insertOne({name="LXZ"}, {name="LXZ", password="1", email="lixizhi@yeah.net
↪"}, function(err, data)  assert(data.password=="1")   end)

    -- insert another one
    db.User:insertOne({name="LXZ2"}, {name="LXZ2", password="123", email=
↪"lixizhi@yeah.net"}, function(err, data)  assert(data.password=="123")     end)
    -- update one
    db.User:updateOne({name="LXZ2",}, {name="LXZ2", password="2", email="lixizhi@yeah.
↪net"}, function(err, data)  assert(data.password=="2") end)
    -- remove and update fields
    db.User:updateOne({name="LXZ2",}, {_unset = {"password"}, updated="with unset"},␣
↪function(err, data)  assert(data.password==nil and data.updated=="with unset") end)
    -- replace the entire document
    db.User:replaceOne({name="LXZ2",}, {name="LXZ2", email="lixizhi@yeah.net"},␣
↪function(err, data)  assert(data.updated==nil) end)
    -- force flush to disk, otherwise the db IO thread will do this at fixed interval
    db.User:flush({}, function(err, bFlushed) assert(bFlushed==true) end);
    -- select one, this will automatically create `name` index
    db.User:findOne({name="LXZ"}, function(err, user) assert(user.password=="1");  ␣
↪end)
    -- array field such as {"password", "1"} are additional checks, but does not use␣
↪index.
    db.User:findOne({name="LXZ", {"password", "1"}, {"email", "lixizhi@yeah.net"}},␣
↪function(err, user) assert(user.password=="1"); end)
    -- search on non-unqiue-indexed rows, this will create index `email` (not-unique␣
↪index)
    db.User:find({email="lixizhi@yeah.net"}, function(err, rows) assert(#rows==2);␣
↪end);
    -- search and filter result with password=="1"
    db.User:find({name="LXZ", email="lixizhi@yeah.net", {"password", "1"}, },␣
↪function(err, rows) assert(#rows==1 and rows[1].password=="1"); end);
    -- find all rows with custom timeout 1 second
    db.User:find({}, function(err, rows) assert(#rows==4); end, 1000);
    -- remove item
    db.User:deleteOne({name="LXZ2"}, function(err, count) assert(count==1); end);
    -- wait flush may take up to 3 seconds
    db.User:waitflush({}, function(err, bFlushed) assert(bFlushed==true) end);
    -- set cache to 2000KB
    db.User:exec({CacheSize=-2000}, function(err, data) end);
    -- run select command from Collection
    db.User:exec("Select * from Collection", function(err, rows) assert(#rows==3)␣
↪end);
    -- remove index fields
    db.User:removeIndex({"email", "name"}, function(err, bSucceed) assert(bSucceed ==␣
↪true) end)
    -- full table scan without using index by query with array items.
    db.User:find({ {"name", "LXZ"}, {"password", "1"} }, function(err, rows) assert(
↪#rows==1 and rows[1].name=="LXZ"); end);
    -- find with left subset of previously created compound key "+name+email"
    db.User:find({ ["+name"] = {"1", limit=2} }, function(err, rows) assert(#rows==2);
↪ end);
    -- return at most 1 row whose id is greater than -1
    db.User:find({ _id = { gt = -1, limit = 1, skip == 1} }, function(err, rows)␣
↪assert(#rows==1); echo("all tests succeed!") end);
```

Why A New Database System?

Current SQL/NoSQL implementation can not satisfy following requirements at the same time.

- Keeping data close to computation, much like our brain.
- Store arbitrary data without schema.
- Automatic indexing based on query usage.
- Provide both blocking/non-blocking API.
- Multithreaded architecture without using network connections for maximum local performance.
- Capable of storing hundreds of Giga Bytes of data locally.
- Native document storage format for NPL tables.
- Super easy client API just like manipulating standard NPL/lua tables.
- Easy to setup and deploy with NPL runtime.
- No server configuration, calling client API will automatically start the server on first use.

About Transactions

Each `write/insert` operation is by default a write command (virtual transaction). We will periodically (default is 50ms, see `Store.AutoFlushInterval`) flush all queued commands into disk. Everything during these period will either succeed or fail. If you are worried about data lose, you can manually invoke `flush` command, however doing so will greatly compromise performance. Please note `flush` command will affect the overall throughput of the entire DB system. In general, you can only get about `20-100` flush(real transactions) per second. Without enforcing transaction on each command, you can easily get a throughput of `6000` write commands per second (i.e. could be `100 times` faster).

> There is one solution to get both high throughput and transaction.

The following solution give you ACID of a single command. After issuing a really important group of commands, and you want to ensure that these commands are actually successful like a transaction, the client can issue a `waitflush` command to check if the previous commands are successful. Please note that `waitflush` command may take up to 50ms or `Store.AutoFlushInterval` to return. You can make all calls asynchronous, so 99.99% times user get a fast feed back, but there is a very low chance that user may be informed of failure after 50ms. On client side, you may also need to prevent user to issue next transaction in 50ms,In most cases, users do not click mouse that quick, so this hidden logic goes barely noticed.

The limitation of the above approach is that you can only know whether a group of commands has been successfully flushed to disk, but you can not rollback your changes, nor can you know which commands fail and which are successful in case of multiple commands.

Opinion on Software Achitecture

Like the brain, we recommend that each computer manages its own chunk of data. As modern computers are having more computing cores, it is possible for a single (virtual) machine to manage 100-500GB of data locally or 1000 requests per second. Using a local database engine is the best choice in such situation for both performance and ease of deployment.

To scale up to even more data and requests, we devide data with machines and use higher level programming logics for communications. In this way, we control all the logics with our own code, rather than using general-purpose solutions like memcached, MangoDB(NoSQL), or SQLServer, etc.

Implementation Details

- Each data table(collections of data) is stored in a single sqlite database file.
- Each database file contains a indexed mapping from object-id to object-table(document).
- Each database file contains a schema table telling additional index keys used.
- Each database file contains a key to object-id table for each custom index key.
- All DB IO operations are performaned in a single dedicated NPL thread.

The above general idea can also be found in commercial database engine like CortexDB, see below.

User Guide

Because Table database is schema-less. It requires some caution of the programmer during development.

Auto Key Index

By default, when you are inserting records into the database, it will have only one internal unique key called `_id`. All subsequent query commands such as `findOne` or `find` or `insertOne`, `deleteOne`, `updateOne` command with query fields, such as `name` will automatically add a new index key with corresponding name such as `name`, and rebuild the index table for all existing records. Please note, all indices are NOT-constraint to be unique, it is up to the caller to ensure index usage. Different records with the same key value are all stored in the index table.

> One can also force not-to-use index when querying by using array items rather than key,value pairs in the query parameter, see below.

```
    NPL.load("(gl)script/ide/System/Database/TableDatabase.lua");
    local TableDatabase = commonlib.gettable("System.Database.TableDatabase");
    -- this will start both db client and db server if not.
    local db = TableDatabase:new():connect("temp/mydatabase/", function() end);

    -- Note: `db.User` will automatically create the `User` collection table if not.
    -- clear all data
    db.User:makeEmpty({}, function(err, count) echo("deleted"..(count or 0)) end);
    -- insert 1
    db.User:insertOne(nil, {name="1", email="1@1",}, function(err, data)  echo(data) ␣
↪   end)
    -- insert 1 with duplicate name
    db.User:insertOne(nil, {name="1", email="1@1.dup",}, function(err, data) ␣
↪echo(data)    end)

    -- find or findOne will automatically create index on `name` and `email` field.
    -- indices are NOT forced to be unique. The caller needs to ensure this see␣
↪`insertOne` below.
    -- this will return two records using one index table `name`
    db.User:find({name="1",}, function(err, rows) echo(rows); end);
    -- this will return one record using two index table `name` and `email`
    db.User:find({name="1", email="1@1"}, function(err, rows) echo(rows); end);
    -- force insert, insertOne with no query parameter will force insertion.
    db.User:insertOne(nil, {name="LXZ", password="123"}, function(err, data) ␣
↪echo(data)     end)
    -- this is an update or insert command, if the query has result, it will actually␣
↪update first matching row rather than inserting one.
    -- this is usually a good way to force uniqueness on key or compound keys
    -- so the following will update existing record rather than inserting a new one
    db.User:insertOne({name="LXZ"}, {name="LXZ", password="1", email="lixizhi@yeah.net
↪"}, function(err, data)  echo(data)   end)
```

All query fields in `findOne`, `updateOne`, etc can contain array items, which performs additional filter checks without automatically create index, like below. `name` is an indexed key, where as `password` and `email` are not, so they need be specified as array items in the query parameter.

```
    -- array fields such as {"password", "1"} are additional checks, but does not use␣
↪index.
    db.User:findOne({name="LXZ", {"password", "1"}, {"email", "lixizhi@yeah.net"}},␣
↪function(err, user) echo(user); end)
```

In case, you make a mistake with index during development, please use `DB manager` in NPL code wiki to remove any index that you accidentally create. or simply call `removeIndex` to remove all or given index keys.

Force Key Uniqueness and `Upsert`

If you want to force uniqueness on a given key, one can specify the unique key in query field when inserting a new record, which turns the `insertOne` command into an `Upsert` command. See below:

```
-- this is an update or insert command, if the query has result, it will actually␣
↪update first matching row rather than inserting one.
-- this is usually a good way to force uniqueness on key or compound keys
-- so the following will update existing record rather than inserting a new one
db.User:insertOne({name="LXZ"}, {name="LXZ", password="1", email="lixizhi@yeah.net"},␣
↪function(err, data)  echo(data)   end)
```

Some notes:

- Query commands such as `find`, `findOne`, `insertOne` will automatically create index for fields in query parameter, unless field is specified in array format.
- Query with mixed index and non-index is done by fetching using index first and then filter result with non-indexed fields. If there is no indexed fields, then non-indexed query can be very slow, since it will linearly search on all records, which can be very slow.
- Query with multiple fields is supported. It will first fetch ids using each index field, and calculate the shared ids and then fetch these rows in a batch.

Write-Ahead-Log and Checkpointing

By default, TableDatabase uses write-ahead-log. We will flush data to WAL file every 50ms; and we will do WAL checkpoint every 5 seconds or 1000 dirty pages. One can configure these two intervals, like below. However, it is not recommended to change these values.

```
NPL.load("(gl)script/ide/System/Database/Store.lua");
local Store = commonlib.gettable("System.Database.Store");
-- We will wait for this many milliseconds when meeting the first non-queued command␣
↪before commiting to disk. So if there are many commits in quick succession, it will␣
↪not be IO bound.
```

```
Store.AutoFlushInterval = 50;
Store.AutoCheckPointInterval = 5000;
```

Checkpointing may take 10ms-1000ms depending on usage, during which time it will block all write operations, so one may experience a latency in API once in a while.

Message Queue Size

One can set the message queue size for both the calling thread and db processor thread. The default value is good enough for most situations.

```
db.name:exec({QueueSize=10001});
```

Memory Cache Size

Change the suggested maximum number of database disk pages that TableDatabase will hold in memory at once per open database file. The default suggested cache size is -2000, which means the cache size is limited to 2048KB of memory. You can set it to a much bigger value, since it is only allocated on demand. Negative value is KB, Positive is number of pages, each page is 4KB.

```
db.name:exec({CacheSize=-2000});
```

Bulk Operations and Message Deliver Guarantee

Table database usually has a high throughout such as 5000-40000 requests/sec. However, if you are doing bulk insertion of millions of records. You should do chunk by chunk using callbacks, otherwise you will either reach the limit of NPL thread buffer or TCP socket buffer(in case it is remote process). A good way is to keep at most 1000 active requests at any one time (`System.Concurrency` has helper class for this kind of jobs). This way you can get a good throughput with message delivery guarantees.

Following is example of inserting 1 million records (takes about 23 seconds). You can paste following code to NPL Code Wiki's console to test.

```
    NPL.load("(gl)script/ide/System/Database/TableDatabase.lua");
    local TableDatabase = commonlib.gettable("System.Database.TableDatabase");

        -- this will start both db client and db server if not.
    local db = TableDatabase:new():connect("temp/mydatabase/");

    db.insertNoIndex:makeEmpty({});
    db.insertNoIndex:flush({});

    NPL.load("(gl)script/ide/Debugger/NPLProfiler.lua");
    local npl_profiler = commonlib.gettable("commonlib.npl_profiler");
    npl_profiler.perf_reset();

    npl_profiler.perf_begin("tableDB_BlockingAPILatency", true)
    local total_times = 1000000; -- a million non-indexed insert operation
    local max_jobs = 1000; -- concurrent jobs count
    NPL.load("(gl)script/ide/System/Concurrent/Parallel.lua");
    local Parallel = commonlib.gettable("System.Concurrent.Parallel");
    local p = Parallel:new():init()
    p:RunManyTimes(function(count)
        db.insertNoIndex:insertOne(nil, {count=count, data=math.random()},␣
↪function(err, data)
            if(err) then
                echo({err, data});
            end
            p:Next();
        end)
    end, total_times, max_jobs):OnFinished(function(total)
        npl_profiler.perf_end("tableDB_BlockingAPILatency", true)
        log(commonlib.serialize(npl_profiler.perf_get(), true));
    end);
```

In future version, we may support build-in Bulk API.

Async vs Sync API

Table database provides both sync and asynchronous API, and they can be use simultaneously. However, sync interface is disabled by default. One has to manually enable it, such as during initialization. In sync interface, if you do not provide a callback function, then the API block until result is returned, otherwise the API return AsyncTask object immediately. See following example:

```
-- enable sync mode once and for all in current thread.
db:EnableSyncMode(true);
-- synchronous call will block until data is fetched.
local err, data = db.User:insertOne(nil, {name="LXZ2", email="sync mode"})
-- async call return a task object immediately without waiting result.
local task = db.User:insertOne(nil, {name="LXZ2", email="sync mode"}, function(err,␣
↪data) end)
```

Synchronous call will pause the entire NPL thread, and is `NOT` the recommended to use, that is why we disabled this feature by default.

Instead, during web development, there is another way to use async API in synchronous way, which is to use coroutine. In NPL web server, we provide `resume/yield`. See [[NPLServerPage]]'s mixing sync/async code section.

Ranged Query and Pagination

Paging through your data is one of the most common operations with TableDB. A typical scenario is the need to display your results in chunks in your UI. If you are batch processing your data it is also important to get your paging strategy correct so that your data processing can scale.

Let's walk through an example to see the different ways of paging through data in TableDB. In this example, we have a database of user data that we need to page through and display 10 users at a time. So in effect, our page size is 10. Let us first create our db with 10000 users of scheme `{_id, name, company, state}`.

```
    NPL.load("(gl)script/ide/System/Database/TableDatabase.lua");
    local TableDatabase = commonlib.gettable("System.Database.TableDatabase");
    local db = TableDatabase:new():connect("temp/mydatabase/");

    -- add some data
    for i=1, 10000 do
        db.pagedUsers:insertOne({name="name"..i},
            {name="name"..i, company="company"..i, state="state"..i}, function() end)
    end
```

Approach One: Use `limit` and `skip`

TableDB support `limit` and `offset|skip` key words in query parameter on an indexed field or the internal `_id` field. `offset or skip` tells TableDB to skip some items before returning at most `limit` number of items. `gt` means `greater than` to select a given range of data.

```
    -- page 1
    db.pagedUsers:find({_id = {gt=-1, limit=10, skip=0}}, function(err, users) ␣
↪echo(users) end);
    -- page 2
    db.pagedUsers:find({_id = {gt=-1, limit=10, skip=10}}, function(err, users) ␣
↪echo(users) end);
    -- page 3
    db.pagedUsers:find({_id = {gt=-1, limit=10, skip=20}}, function(err, users) ␣
↪echo(users) end);
```

You get the idea. In general to retrieve page `n` the code looks like this

```
    local pagesize = 10; local n = 10;
    db.pagedUsers:find({_id = {gt=-1, limit=pagesize, skip=(n-1)*pagesize}},␣
↪function(err, users)  echo(users) end);
```

However as the `offset` of your data increases this approach has serious performance problems. The reason is that every time the query is executed, the server has to walk from the beginning of the first non-skipped row to the specified offset. As your offset increases this process gets slower and slower. Also this process does not make efficient use of the indexes. So typically the `skip` and `limit` approach is useful when you have small `offset`. If you are working with large data sets with very big `offset` values you need to consider other approaches.

Approach Two: Using `find` and `limit`

The reason the previous approach does not scale very well is the `skip` command. So the goal in this section is to implement paging without using the `skip` command. For this we are going to leverage the natural order in the stored data like a time stamp or an id stored in the document. In this example we are going to use the `_id` stored in each document. `_id` is a TableDB auto-increased internal id.

```
    -- page 1
    db.pagedUsers:find({_id = { gt = -1, limit=10}}, function(err, users)
        echo(users)
        -- Find the id of the last document in this page
        local last_id = users[#users]._id;

        -- page 2
        db.pagedUsers:find({_id = { gt = last_id, limit=10}}, function(err, users)
           echo(users)
           -- Find the id of the last document in this page
           local last_id = users[#users]._id;
           -- page 3
           -- ...
        end);
    end);
```

Notice how the `gt` keywords to select rows whose values is `greater than` the specified value and sort data in ascending order. One can do this for any indexed field, either it is number or string, see below for ranged query in TableDB:

```
NPL.load("(gl)script/ide/System/Database/TableDatabase.lua");
local TableDatabase = commonlib.gettable("System.Database.TableDatabase");

-- this will start both db client and db server if not.
local db = TableDatabase:new():connect("temp/mydatabase/");

-- add some data
for i=1, 100 do
    db.rangedTest:insertOne({i=i}, {i=i, data="data"..i}, function() end)
end

-- return at most 5 records with i > 90, skipping 2. result in ascending order
db.rangedTest:find({ i = { gt = 95, limit = 5, offset=2} }, function(err, rows)
    echo(rows); --> 98,99,100
end);

-- return at most 20 records with _id > 98, result in ascending order
db.rangedTest:find({ _id = { gt = 98, limit = 20} }, function(err, rows)
    echo(rows); --> 99,100
end);

-- do a full table scan without using index
db.rangedTest:find({ {"i", { gt = 55, limit=2, offset=1} }, {"data", {lt="data60"}
↪ }, }, function(err, rows)
    echo(rows); --> 57,58
end);

db.rangedTest:find({ {"i", { gt = 55} }, {"data", "data60"}, }, function(err,␣
↪rows)
    echo(rows); --> 60
end);
```

Please note that array fields in query object will not auto-create or use any index, instead a full table scan is performed which can be slow. Ranged query is also supported in array fields, but it is implemented via full table scan.

Compound key is more suitable for ranged query see Compound keys section below for how to use it.

Count Record

Rule of thumb: avoid using `count`!

Implementation of full paging usually requires you to get the count of all items in a database. However, it is not trivial to get accurate count. SQL query like `select count(*) as count from Collection` may take seconds or even minutes to return for tables with many rows like 100 millions. This is because almost every SQL engine uses B-tree, but B-tree does not store count. The same story is true for both TableDB, MySQL, etc.

For a faster count, you can create a counter table and let your application update it according to the inserts and deletes it does. Or you may not allow the user to scroll to the last page, but show more pages progressively.

If you do not mind the performance or your table size is small, TableDB provides the `count` query

```
NPL.load("(gl)script/ide/System/Database/TableDatabase.lua");
local TableDatabase = commonlib.gettable("System.Database.TableDatabase");
local db = TableDatabase:new():connect("temp/mydatabase/");

db.countTest:removeIndex({}, function(err, bRemoved) end);
```

```
    -- compound keys
    for i=1, 100 do
        db.countTest:insertOne({name="name"..i}, {
            name="name"..i,
            company = (i%2 == 0) and "tatfook" or "paraengine",
            state = (i%3 == 0) and "china" or "usa"}, function() end)
    end

    -- count all rows
    db.countTest:count({}, function(err, count)
        assert(count == 100)
    end);
    -- count with compound keys
    db.countTest:count({["+company"] = {"tatfook"}}, function(err, count)
        assert(count == 50)
    end);
    -- count with complex query
    db.countTest:count({["+state+name+company"] = {"china", gt="name50"}},␣
↪function(err, count)
        assert(count == 19)
    end);
```

Compound Indices

To efficiently query data with multiple key values, one needs to use compound indices. Please note a compound key of (key1, key2, key3) can also be used to query (key1, key2) and (key1), so the order of subkeys are very important.

If you created a compound index say on (key1, key2, key3), then only the right most key in the query can contain ranged constraint, all other keys to its left such as (key1 and key2) must be present (no gaps) and specified in equal constraint.

To understand compound indices, one needs to understand how index works in standard relational database engine, because a TableDB query usually use only one internal index to do all the complex queries and an index is a B-tree. So compound index can not be used where B-tree is not capable of. Read query planner of sqlite here for more details.

The syntax of using compound key is by prepending "+|-" to one or more field names, such as "+key1+key2+key3", there can be at most 4 sub keys, see below for examples. "+" means ascending order, "-" means descending order.

Examples:

```
    NPL.load("(gl)script/ide/System/Database/TableDatabase.lua");
    local TableDatabase = commonlib.gettable("System.Database.TableDatabase");
    local db = TableDatabase:new():connect("temp/mydatabase/");

    db.compoundTest:removeIndex({}, function(err, bRemoved) end);

    -- compound keys
    for i=1, 100 do
        db.compoundTest:insertOne({name="name"..i}, {
            name="name"..i,
            company = (i%2 == 0) and "tatfook" or "paraengine",
            state = (i%3 == 0) and "china" or "usa"}, function() end)
    end

    -- compound key can also be created on single field "+company". it differs from␣
↪standard key "company".
    db.compoundTest:find({["+company"] = {"paraengine", limit=5, skip=3}},␣
↪function(err, users)
```

```
        assert(#users == 5)
    end);

    -- create a compound key on +company+name (only the right most key can be paged)
    -- `+` means index should use ascending order, `-` means descending. such as
↪"+company-name"
    -- it will remove "+company" key, since the new component key already contains␣
↪the "+company".
    db.compoundTest:find({["+company+name"] = {"tatfook", gt="", limit=5, skip=3}},␣
↪function(err, users)
        assert(#users == 5)
    end);

    -- a query of exact same left keys after a compound key is created will␣
↪automatically use
    -- the previously created compound key, so "+company",  "+company+name" in␣
↪following query are the same.
    db.compoundTest:find({["+company"] = {"tatfook", limit=5, skip=3}}, function(err,␣
↪users)
        assert(#users == 5)
    end);

    -- the order of key names in compound index is important. for example:
    -- "+company+state+name" shares the same compound index with "+company" and
↪"+company+state"
    -- "+state+name+company" shares the same compound index with "+state" and
↪"+state+name"
    db.compoundTest:find({["+state+name+company"] = {"china", gt="name50", limit=5,␣
↪skip=3}}, function(err, users)
        assert(#users == 5)
    end);

    -- compound keys with descending order. Notice the "-" sign before `name`.
    db.compoundTest:find({["+state-name+company"] = {"china", limit=5, skip=3}},␣
↪function(err, users)
        assert(#users == 5)
    end);

    -- updateOne with compound key
    db.compoundTest:updateOne({["+state-name+company"] = {"usa", "name1", "paraengine
↪"}}, {name="name0_modified"}, function(err, user)
        assert(user.name == "name0_modified")
    end);

    -- this query is ineffient since it uses intersection of single keys (with many␣
↪duplications).
    -- one should consider use "+state+company", instead of this.
    db.compoundTest:find({state="china", company="tatfook"}, function(err, users)
        assert(#users == 16)
    end);

    -- this query is ineffient since it uses intersection of single keys.
    -- one should consider use "+state+company", instead of this.
    db.compoundTest:count({state="china", company="tatfook"}, function(err, count)
        assert(count == 16)
    end);
```

Compound key vs Standard key

Compound key is very different from standard key in terms of internal implementations. They can coexist in the same table even for the same key. For example:

```
-- query with standard key "name"
db.User:find({name="1",});
-- query with compound key "+name"
db.User:find({["+name"]="1",});
```

The above code will create two different kinds of internal index tables for the "name" field.

Compound key index table is more similar to traditional relational database table. For example, if one creates a compound key of "+key1+key2+key3", then an internal index table of `(cid UNIQUE, name1, name2, name3)` is created with composite index of `CREATE INDEX (name1, name2, name3)`. For each row in the collection, there is a matching row in its internal index table.

However, a standard key in tableDB stores index table differently as `(name UNIQUE, cids)`, where cids is a text array of collection `cid`. There maybe far less rows in the internal index table than the actual collection.

Both keys have advantages and disadvantages, generally speaking:

- standard key is faster and good for index intersections. Most suitable for keys with 1-1000 duplications.
- compound key is good for sorting and ranged query. Suitable for keys with any duplicated rows.

The biggest factor that you should use a compound key is that your key's values have many duplications. For example, if you want to query with `gender+age` with millions of `user` records, compound key is the only way to go.

For other single key situations, standard key is recommended. For example, with standard keys you can get all `projects` that belongs to a given `userid` very efficiently, given that each user has far less than 1000 projects.

Database Viewer Tools

[[NPLCodeWiki]] contains a `db manager` in its tools menu, which can be used to view and modify data, as well as examine and removing `table indices`. It is very important to examine and remove unnecessary indices due to coding mistakes.

Remove Table Fields

To remove table fields, use `updateOne` with `_unset` query parameter like below.

```
    -- remove "password" field from the given row
    db.User:updateOne({name="LXZ2",}, {_unset = {"password"}, updated="with unset"},
       function(err, data)
          assert(data.password==nil and data.updated=="with unset")
    end)
```

Another way is the use `replaceOne` to replace the entire document.

```
-- replace the entire document
db.User:replaceOne({name="LXZ2",}, {name="LXZ2", email="lixizhi@yeah.net"},␣
↪function(err, data)  assert(data.updated==nil) end)
```

Always Use Models

In typical web applications, we use model/view/controller (or MVC) architecture. A model is usually a wrapper of a single schema-less database record in TableDatabase. Usually a model should provide CRUD(create/read/update/delete) operations according to user-defined query. It is the model's responsibility to ensure input/output are valid, such as whether a string is too long, or whether the caller is querying with a non-indexed key.

In case, you are using client side controllers, one can use a router page to automatically redirect URL AJAX queries to the model's function calls.

For a complete example of url redirection and model class, please see `script/apps/WebServer/admin/wp-content/pages/wiki/routes.page` and `script/apps/WebServer/admin/wp-content/pages/models/abstract/base.page`

Following code illustrates the basic ideas.

Router page such as in `routes.page`

```
        local model = models and models[modelname];
        if(not model) then
            return response:status(404):send({message="model not found"});
        else
            model = model:new();
        end

        -- redirect CRUB URL to method in model.
        if(request:GetMethod() == "GET") then
            if(model.get) then
                local result = model:get(request:getparams());
                return response:send(result);
            end
        elseif(request:GetMethod() == "PUT") then
            if(params and params:match("^new")) then
                if(model.create) then
                    local result = model:create(request:getparams());
                    return response:send(result);
                end
            else
                if(model.update) then
                    local result = model:update(request:getparams());
                    return response:send(result);
                end
            end
        elseif(request:GetMethod() == "DELETE") then
            if(model.delete) then
                local result = model:delete(request:getparams());
                return response:send(result);
            end
        end
```

And in model class, such as `models/site.page`

```
<?npl
--[[
Title: a single web site of a user
Author: LiXizhi
Date: 2016/6/28
]]
```

```
include_once("base.page");

local site = inherit(models.base, gettable("models.site"));

site.db_name = "site";

function site:ctor()
    -- unique name of the website, usually same as owner name
    self:addfield("name", "string", true, 30);
    -- markdown text description
    self:addfield("desc", "string");
    -- such as "https://github.com/LiXizhi/wiki"
    self:addfield("store", "string", false, 200);
    -- owner name, not unique
    self:addfield("owner", "string", false, 30);
end
```

Data Replication

TableDB is a fully ACID and fast database (it uses Sqlite as its internal storage engine). We are going to support tableDB very actively as an integral part of NPL. NPL language itself is designed for parallelism, the current version of tableDB is still a local SQL engine. It is hence not hard to use NPL to leverage TableDB to a fully distributed and fault-tolerant database system.

TODO: RAFT implementation coming in 2017

[[https://raft.github.io/]] is a concensus algorithm that we are going to implement using NPL and TableDB itself to leverage TableDB to support `master/slave` data replication for high-availability database systems.

There are many existing distributed SQL server that are built over similar architecture and use Sqlite as the storage engine, like these

- [[http://www.actordb.com/]]
- [[http://www.cortex-ag.com/]]
- [[https://github.com/rqlite/rqlite]]

  We highly recommend the native database system in NPL: [[TableDatabase|UsingTableDatabase]]. It is way faster and built-in with NPL runtime.

CHAPTER 55

Using MySql Client

Install Guide

- On linux platform, mysql client is by default installed when you build NPLRuntime from source code. You should find a `libluasql.so` in NPL runtime directory.
- On windows platform, you must manually install the mysql client connector plugin.

  [[https://github.com/LiXizhi/luasql]]

Once installed, you can use it like below:

```
NPL.load("(gl)script/ide/mysql/mysql.lua");
local MySql = commonlib.gettable("System.Database.MySql");
local mysql_db = MySql:new():init(db_user, db_password, db_name, db_host, db_port);
```

A Better Way to Code

It is recommended that you write a wrapper file like `my_db.page` which exposes a global object such as `my_db` that contains functions to access to your actual database.

An example in the Admin web site framework is here.

- `script/apps/WebServer/admin/wp-includes/wp-db.page`

It will manage connection strings (passwords, ips, pools) from global configuration files, etc. In all other server pages, you simply include the file, and call its functions like this

```
local query = string.format("select * from wp_users where %s = '%s' ",db_field,value);
local user = wpdb:get_row(query);
```

CHAPTER 56

Deploy NPL Web Server With SSL (https)

This post shows `How To Configure Nginx with SSL as a Reverse Proxy for NPL Web Server`.

Why NPL Does Not Support SSL Natively?

NPL protocol is TCP based, which can run HTTP and NPL's TCP protocol on the `same` port. If the TCP connection is encoded with SSL, it will break NPL's internal TCP protocol. This is why NPL does not support SSL natively. However, NPL server can fetch data via `https`, so it is perfectly fine to call SSL protected rest API within the NPL server.

Introduction

By default, NPL comes with its own built in web server, which listens on port 8099. This is convenient if you run a private NPL instance, or if you just need to get something up quickly and don't care about security. Once you have real production data going to your host, though, it's a good idea to use a more secure web server proxy in front like Nginx.

Prerequisites

This post will detail how to wrap your site with SSL using the Nginx web server as a reverse proxy for your NPL instance. This tutorial assumes some familiarity with Linux commands, a working NPL Runtime installation, and a Ubuntu 14.04 installation.

You can install NPL runtime later in this tutorial, if you don't have it installed yet.

Step One — Configure Nginx

Nginx has become a favored web server for its speed and flexibility in recents years, so that is the web server we will be using.

Install Nginx

Update your package lists and install Nginx:

```
sudo apt-get update
sudo apt-get install nginx
```

Get a Certificate

Next, you will need to purchase or create an SSL certificate. These commands are for a self-signed certificate, but you should get an officially signed certificate if you want to avoid browser warnings.

Move into the proper directory and generate a certificate:

```
cd /etc/nginx
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/nginx/cert.key -
↪out /etc/nginx/cert.crt
```

You will be prompted to enter some information about the certificate. You can fill this out however you'd like; just be aware the information will be visible in the certificate properties. We've set the number of bits to 2048 since that's the minimum needed to get it signed by a CA. If you want to get the certificate signed, you will need to create a CSR.

Edit the Configuration

Next you will need to edit the default Nginx configuration file.

```
sudo nano /etc/nginx/sites-enabled/default
```

Here is what the final config might look like; the sections are broken down and briefly explained below. You can update or replace the existing config file, although you may want to make a quick copy first.

```
server {
    listen 80;
    return 301 https://$host$request_uri;
}

server {

    listen 443;
    server_name npl.domain.com;

    ssl_certificate           /etc/nginx/cert.crt;
    ssl_certificate_key       /etc/nginx/cert.key;

    ssl on;
    ssl_session_cache  builtin:1000  shared:SSL:10m;
    ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers HIGH:!aNULL:!eNULL:!EXPORT:!CAMELLIA:!DES:!MD5:!PSK:!RC4;
    ssl_prefer_server_ciphers on;

    access_log            /var/log/nginx/npl.access.log;

    location / {

      proxy_set_header        Host $host;
      proxy_set_header        X-Real-IP $remote_addr;
      proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header        X-Forwarded-Proto $scheme;

      # Fix the "It appears that your reverse proxy set up is broken" error.
      proxy_pass          http://localhost:8099;
      proxy_read_timeout  90;

      proxy_redirect      http://localhost:8099 https://npl.domain.com;
    }
  }
```

In our configuration, the cert.crt and cert.key settings reflect the location where we created our SSL certificate. You will need to update the servername and `proxyredirect` lines with your own domain name. There is some additional Nginx magic going on as well that tells requests to be read by Nginx and rewritten on the response side to ensure the reverse proxy is working.

The first section tells the Nginx server to listen to any requests that come in on port 80 (default HTTP) and redirect them to HTTPS.

```
...
server {
   listen 80;
   return 301 https://$host$request_uri;
}
...
```

Next we have the SSL settings. This is a good set of defaults but can definitely be expanded on. For more explanation, please read this tutorial.

```
...
  listen 443;
  server_name npl.domain.com;

  ssl_certificate           /etc/nginx/cert.crt;
  ssl_certificate_key       /etc/nginx/cert.key;

  ssl on;
  ssl_session_cache  builtin:1000  shared:SSL:10m;
  ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
  ssl_ciphers HIGH:!aNULL:!eNULL:!EXPORT:!CAMELLIA:!DES:!MD5:!PSK:!RC4;
  ssl_prefer_server_ciphers on;
  ...
```

The final section is where the proxying happens. It basically takes any incoming requests and proxies them to the NPL instance that is bound/listening to port 8099 on the local network interface. This is a slightly different situation, but this tutorial has some good information about the Nginx proxy settings.

```
...
location / {
    proxy_set_header        Host $host;
    proxy_set_header        X-Real-IP $remote_addr;
    proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header        X-Forwarded-Proto $scheme;

    # Fix the "It appears that your reverse proxy set up is broken" error.
    proxy_pass          http://localhost:8099;
    proxy_read_timeout  90;

    proxy_redirect      http://localhost:8099 https://npl.domain.com;
}
...
```

A few quick things to point out here. If you don't have a domain name that resolves to your NPL server, then the proxy_redirect statement above won't function correctly without modification, so keep that in mind.

Step Two — Configure NPL Runtime

As stated previously, this tutorial assumes that NPL is already installed. This tutorial will show you how to install NPL if necessary. You will probably need to switch to the root user for that article.

For NPL to work with Nginx, we need to update the NPL config to listen only on the localhost interface instead of all (0.0.0.0), to ensure traffic gets handled properly. This is an important step because if NPL is still listening on all interfaces, then it will still potentially be accessible via its original port (8099).

For example, one may start npl webserver like this

```
pkill -9 npl
npl -d root="WebServerExample/" ip="127.0.0.1" port="8099" bootstrapper="script/apps/
↪WebServer/WebServer.lua"
```

Notice that the `ip="127.0.0.1"` setting needs to be either added or modified.

Then go ahead and restart NPL and Nginx.

```
sudo service nginx restart
```

You should now be able to visit your domain using either HTTP or HTTPS, and the NPL web site will be served securely. You will see a certificate warning if you used a self-signed certificate.

Conclusion

The only thing left to do is verify that everything worked correctly. As mentioned above, you should now be able to browse to your newly configured URL - npl.domain.com - over either HTTP or HTTPS. You should be redirected to the secure site, and should see some site information, including your newly updated SSL settings. As noted previously, if you are not using hostnames via DNS, then your redirection may not work as desired. In that case, you will need to modify the proxy_pass section in the Nginx config file.

You may also want to use your browser to examine your certificate. You should be able to click the lock to look at the certificate properties from within your browser.

References

This is a rewrite of this post in terms of NPL.

CHAPTER 57

Use Links

Source code

- NPL Runtime
- main package
- paracraft package

Development

- Download Paracraft
- Download ParacraftSDK

Documentation

- NPL Runtime Documentation
- ParacraftSDK Documentation
- Paracraft Wiki

Plugins and Mod

- NPL Cad
- NPL Exporter

CHAPTER 58

NPL Runtime Performance Compare

High-Performance JIT Compiler

NPL syntax is 100% compatible with Lua, therefore it can be configured to utilize JIT compiler (Luajit). In general, luajit is believed to be one of the fastest JIT compiler in the world. It's speed is close to C/C++, and can even outperform static typed languages like java/C#. It is the fastest dynamic language in the world. However, special care needs to be taken when writing test cases, since badly written test case can make the same code 100 times slower.

Compare Chart

Following is from Julia. Source code: C, Fortran, Python, Matlab/Octave, R, JavaScript, Java, Go, Lua.

The following micro-benchmark results were obtained on a single core (serial execution) on an Intel(R) Xeon(R) CPU E7-8850 2.00GHz CPU with 1TB of 1067MHz DDR3 RAM, running Linux:

	C/Fortran	Python	**NPL**	Java	JavaScript	Go	R	Matlab	Octave
	gcc5.1	3.4.3	1.0	1.8	V8	go1.5	3.2.2	R2015b	4.0.0
mandel	0.81	15.32	**0.67**	1.35	0.66	1.11	53.16	7.58	451.81
fib	0.70	77.76	**1.71**	1.21	3.36	1.86	533.52	26.89	9324.35
rand_mat_mul	3.48	1.14	**1.16**	2.36	15.07	1.42	1.57	1.12	1.12
rand_mat_stat	1.45	17.93	**3.27**	3.92	2.30	2.96	14.56	14.52	30.93
parse_int	5.05	17.02	**5.77**	3.35	6.06	1.20	45.73	802.52	9581.44
quicksort	1.31	32.89	**2.03**	2.60	2.70	1.29	264.54	4.92	1866.01
pi_sum	1.00	21.99	**1.00**	1.00	1.01	1.00	9.56	1.00	299.31

Figure: benchmark times relative to C (smaller is better, C performance = 1.0).

> C and Fortran compiled by gcc 5.1.1, taking best timing from all optimization levels (-O0 through -O3). C, Fortran, Go. Python 3 was installed from the Anaconda distribution. The Python implementations of rand_mat_stat and rand_mat_mul use NumPy (v1.9.2) functions; the rest are pure Python implementations. Benchmarks can also be seen here as a plot created with Gadfly.

These benchmarks, while not comprehensive, do test compiler performance on a range of common code patterns, such as function calls, string parsing, sorting, numerical loops, random number generation, and array operations. It is important to note that these benchmark implementations are not written for absolute maximal performance (the fastest code to compute fib(20) is the constant literal 6765). Rather, all of the benchmarks are written to test the performance of specific algorithms implemented in each language. In particular, all languages use the same algorithm:

the Fibonacci benchmarks are all recursive while the pi summation benchmarks are all iterative; the "algorithm" for random matrix multiplication is to call LAPACK, except where that's not possible, such as in JavaScript. The point of these benchmarks is to compare the performance of specific algorithms across language implementations, not to compare the fastest means of computing a result, which in most high-level languages relies on calling C code.

NPL Database Performance

More information, please see [[UsingTableDatabase]]

Following is tested on my Intel-i7-3GHZ CPU. See test folder for test source code.

Run With Conservative Mode

Following is averaged value from 100000+ operations in a single thread

- Random non-index insert: `43478 inserts/second`
  - Async API tested with default configuration with 1 million records on a single thread.
- Round trip latency call:
  - Blocking API: `20000 query/s`
  - Non-blocking API: `11ms` or `85 query/s` (due to NPL time slice)
  - i.e. Round strip means start next operation after previous one is returned. This is latency test.
- Random indexed inserts: `17953 query/s`
  - i.e. start next operation immediately, without waiting for the first one to return.
- Random select with auto-index: `18761 query/s`
  - i.e. same as above, but with findOne operation.
- Randomly mixing CRUD operations: `965-7518` query/s
  - i.e. same as above, but randomly calling Create/Read/Update/Delete (CRUD) on the same auto-indexed table.
  - Mixing read/write can be slow when database grows bigger. e.g. you can get `18000 CRUD/s` for just 10000 records.

NPL Web Server Performance

The following is done using ApacheBench (AB tool) with 5000 requests and 1000 concurrency on a 1 CPU 1GB memory virtual machine. The queried content is [[http://paracraft.wiki/]], which is a fairly standard dynamic web page.

```
root@iZ62yqw316fZ:/opt/paracraftwiki# ab -n 5000 -c 1000 http://paracraft.wiki/
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking paracraft.wiki (be patient).....done

Finished 5000 requests
```

```
Server Software:        NPL/1.1
Server Hostname:        paracraft.wiki
Server Port:            80

Document Path:          /
Document Length:        24736 bytes

Concurrency Level:      1000
Time taken for tests:   3.005 seconds
Complete requests:      5000
Failed requests:        0
Write errors:           0
Total transferred:      124730000 bytes
HTML transferred:       123680000 bytes
Requests per second:    1664.05 [#/sec] (mean)
Time per request:       600.944 [ms] (mean)
Time per request:       0.601 [ms] (mean, across all concurrent requests)
Transfer rate:          40538.43 [Kbytes/sec] received
```

References

There are a few benchmark compare sites:

- [[https://github.com/attractivechaos/plb]]: [[algorithm compare |http://attractivechaos.github.io/plb/]]
- TechEmpower: compare web server framework only
- Luajit Performance
- Computer benchmark game: No luajit.

CHAPTER 59

ParaEngine

namespace **`ParaEngine`**

Functions

int **`my_rand`**()

class **`IObjectScriptingInterface`**

Public Functions

bool **`AddScriptCallback`**(int *func_type*, **const** string &*script_func*)
add a new call back handler. it will override the previous one if any.
Parameters
- `script_func`: format is [filename];[scode] where file name can be any NPL address, scode is a short code sequence to execute. it may has the same format as the GUI event handler. e.g. ";function1()" : calling a global function "(gl)script/character/npc.lua;npc.on_click()" : load script if not loaded before and then calling a global function if this is "", *RemoveScriptCallback()* will be called.

IObjectScriptingInterface::ScriptCallback ***`GetScriptCallback`**(int *func_type*)
return NULL if there is no associated script.

bool **`RemoveScriptCallback`**(int *func_type*)
remove a call back handler

struct `ScriptCallback`
#include <IObjectScriptingInterface.h> the game object will keep a list of script call backs.

Public Functions

void **`SetScriptFunc`**(**const** std::string &*script*)
set the script function

int **`ActivateAsync`**(**const** std::string &*code*)
activate the script callback with the given code.

int **`ActivateLocalNow`**(**const** std::string &*script*)
activate the script callback locally now. when function returns, the script has returned.

Public Members

int **`func_type`**
function type

std::string **`script_func`**
the NPL file and function to be activated. Its format is similar to GUI events. e.g. "(gl)script/NPC1.lua;NPC1.On_Click();"

unsigned int **`m_nLastTick`**
last time this function is called.

CHAPTER 60

ParaScripting

namespace ParaScripting

for luabind, The main drawback of this approach is that the compilation time will increase for the file that does the registration, it is therefore recommended that you register everything in the same cpp-file.

ParaScripting contains all classes and functions used for communication between the game engine and scripts.

Typedefs

typedef pair<**const** char *, *ParaObjectNode*> **ParaObject_Pair**

Functions

int **NPL_dummy** (lua_State **L*)

split registration to save compiling time. void register_part1(class_<X>& x){ x.def(...); } void register_part2(class_<X>& x){ x.def(...); } void register_(lua_State* L){ class_<X> x("x"); register_part1(x); register_part2(x); module(L) [x]; }this function does nothing but count as one instruction in preemptive function.

static void **lua_register_funcs_totable** (lua_State **L*, *lua_func_entry* *entries*[])

for registering functions

static void **lua_register_values_totable** (lua_State **L*, lua_vale_entry *entries*[])

for registering values

static int **NPL_Print** (lua_State **L*)

e.g. print("Hello ParaEngine %d", 10); Receives any number of arguments, and prints their values in stdout, using the tostring function to convert them to strings. This function is not intended for formatted output, but only as a quick way to show a value, typically for debugging. For formatted output, use string.format

static int **`NPL_Format`** (lua_State **L*)
: this function is the same as string.format, except that it only support d, and s, and f (always printed as %.2f) We do not use sprintf internally, so it runs much faster than string.format which uses sprintf. e.g. a = format("%s hello %d, %f", "Hello ParaEngine",10, 10.1234);

static int **`luaopen_luaxml`** (lua_State **L*)
: the luaxml api will be returned in a table, usually we can assign this table like this *ParaXML* = luaopen_luaxml();

template <typename T**>**
T ***`get_pointer`** (*StackObjectPtr*<T> **const** &*pointer*)

template <typename T**>**
T ***`get_pointer`** (T **const** &*pointer*)
: this defines a rule, so that every object defined in *ParaScripting* namespace will be treated as StackObject by luabind. i.e. all objects such as *ParaObject*, *ParaUIObject*, etc are passed by value on the stack, instead of using std::auto_ptr(new ParaObject(p)). if one wants to pass by reference, overwrite this method.

  See : see make_pointee_instance in luabind/policy.hpp and object_rep.hpp.

string **`FileTimeToDateString`** (**const** FILETIME **pTime*)

template <class T**>**
void **`traverse`** (**const** Json::Value &*var*, **const** object &*outTable*, **const** T &*sKey*, bool *bFirstTable* = false)

OBJECT_FILTER_CALLBACK **`GetFilterFuncByName`** (**const** char **sFilterFunc*)

DWORD **`GetPhysicsGroupMaskByName`** (**const** char **sFilterFunc*)
: get the physics group mask by filter name

Variables

lua_func_entry **`luaxml_api_entries`**[] = { { "LuaXML_ParseFile", ParaXML::LuaXML_ParseFile }, { "LuaXML_ParseSt

CParaFile **`g_currentIOfile`**
: the current IO file.

bool **`s_bIsWhiteSpaceCollapsed`** = true

class `CAssetScriptCallbackData`
: terrain mask file callback data

class `CNPL`
: *#include <ParaScriptingNPL.h>* Neural Parallel Language functions are in this namespace.

Public Static Functions

int **`activate`** (**const** object &*sNPLFilename*, **const** object &*sCode*)
: activate the specified file. It can either be local or remote file.

  the following is a list of all valid file name combinations: "user001@paraengine.com:script/hello.lua" a file of user001 in its default gaming thread "(world1)server001@paraengine.com:script/hello.lua" a file of server001 in its thread world1 "(worker1)script/hello.lua" a local file in the thread worker1 "(gl)script/hello.lua" a glia (local) file in the current runtime state's thread "script/hello.lua" a file in the current thread. For a single threaded application, this is usually enough. "(worker1)NPLRouter.dll" activate a C++ or C# dll. Please note that, in windows, it looks for NPLRonter.dll; in linux, it looks for ./libNPLRouter.so "plugin/libNPLRouter.dll" almost same as above, it is recommented to remove the heading 'lib' when loading. In windows, it looks for plugin/NPLRonter.dll; in linux, it looks for ./plugin/libNPLRouter.so

Parameters

- `pState`: the source runtime state that initiated this activation. If pState is NULL, the main runtime state is used.
- `sNPLFileName`: a globally unique name of a NPL file name instance. The string format of an NPL file name is like below. [(sRuntimeStateName|gl)][sNID:]sRelativePath[]

Note : pure data table is defined as table consisting of only string, number and other table of the above type. NPL.activate function also accepts *ParaFileObject* typed message data type. *ParaFileObject* will be converted to base64 string upon transmission. There are size limit though of 10MB. one can also programmatically check whether a script object is pure date by calling NPL.SerializeToSCode() function. Please note that data types that is not pure data in sCode will be ignored instead of reporting an error.

Return : NPLReturnCode. 0 means succeed.

Parameters

- `code`: it is a chunk of pure data table init code that would be transmitted to the destination file.
- `nLength`: the code length. if this is 0, length is determined from code by finding '\0', but, it must not exceed 4096 bytes. If length is explicitly specified, there is no such a limit.
- `channel:On`: which channel to send the package. It can be a number in [0,15]. In case it is nil, the default channel (0) is used.
- `priority`: From high to low.If this is nil, medium priority(0) is used. following enumerations are used to describe when packets are delivered. enum PacketPriority { SYSTEM_PRIORITY, /// internally Used to send above-high priority messages. HIGH_PRIORITY, /// High priority messages are send before medium priority messages. MEDIUM_PRIORITY, /// Medium priority messages are send before low priority messages. LOW_PRIORITY, /// Low priority messages are only sent when no other messages are waiting. };
- `reliability:From`: unreliable to reliable sequenced. 0 stands for unreliable. If this is nil, RELIABLE_ORDERED(3) is used. following enumerations are used to describe how packets are delivered. enum PacketReliability { UNRELIABLE, /// Same as regular UDP, except that it will also discard duplicate datagrams. It adds (6 to 17) + 21 bits of overhead, 16 of which is used to detect duplicate packets and 6 to 17 of which is used for message length. UNRELIABLE_SEQUENCED, /// Regular UDP with a sequence counter. Out of order messages will be discarded. This adds an additional 13 bits on top what is used for UNRELIABLE. RELIABLE, /// The message is sent reliably, but not necessarily in any order. Same overhead as UNRELIABLE. RELIABLE_ORDERED, /// This message is reliable and will arrive in the order you sent it. Messages will be delayed while waiting for out of order messages. Same overhead as UNRELIABLE_SEQUENCED. RELIABLE_SEQUENCED /// This message is reliable and will arrive in the sequence you sent it. Out or order messages will be dropped. Same overhead as UNRELIABLE_SEQUENCED. };

int **`activate2_`**(**const** char **sNPLFilename*, **const** char **sCode*)

this function is only called by .Net API.

int **`activate_`**(**const** char **sNPLFilename*, **const** char **sCode*, int *channel*, int *priority*, int *reliability*)

this function is only called by .Net API.

void **`call`**(**const** object &*sNPLFilename*, **const** object &*sCode*)

This function is used to activate a local file synchronously in the current runtime state. Use *activate()* if you need an asynchronous activation. for more information, please see *activate()*;

void **`call_`**(**const** char **sNPLFilename*, **const** char **sCode*)

this function is only called by .Net API.

const char ***`GetFileName`**()

return the NPL file name that is being loaded. Only call this function when the file is being initialized. i.e. at the root level. Note: calling this function inside other functions will lead to unexpected result.

void **this_**(**const** object &*funcActivate*)

NOTE: the function name is "this" in NPL, not "this_". associate a user defined function as the activation function of this file. add the current file name to the __act table. create the activate table, if it does not exist.

Parameters

- `funcActivate`: the function pointer to the activation function. It can either be local or global.
- `params`: nil or a table {[PreemptiveCount=number,] [MsgQueueSize=number,] [filename|name=string,]}
  - PreemptiveCount: if PreemptiveCount is omitted (default), the activate function will run non-preemptive (it is the programmer's job to let the function finish in short time). If PreemptiveCount > 0, the activate function will be preemptive (yield) after this number of virtual instructions. which allows us to run tens of thousands of jobs concurrently. Each job has its own stack and but the programmer should pay attention when making changes to shared data.
  - MsgQueueSize: Max message queue size of this file, if not specified it is same as the NPL thread's message queue size.
  - filename|name: virtual filename, if not specified, the current file being loaded is used.
  - clear: clear all memory used by the file, including its message queue. Normally one never needs to clear. A neuron file without messages takes less than 100 bytes of memory (mostly depending on the length's of its filename)

ParaAttributeObject **GetAttributeObject**()

get the attribute object. This function return a clone of this object.

void **load**(**const** object &*filePath*, bool *bReload*)

load a new file (in the current runtime state) without activating it. If the file is already loaded, it will not be loaded again unless bReload is true. IMPORTANT: this function is synchronous; unlike the asynchronous activation function. LoadFile is more like "include in C++".When the function returns, contents in the file is loaded to memory.

Note : in NPL/lua, function is first class object, so loading a file means executing the code chunk in protected mode with pcall, in most cases, this means injecting new code to the global table. Since there may be recursions (such as A load B, while B also load A), your loading code should not reply on the loading order to work. You need to follow basic code injection principles. For example, commonlib.gettable("") is the the commended way to inject new code to the current thread's global table. Be careful not to pollute the global table too much, use nested table/namespace. Different NPL applications may have their own sandbox environments, which have their own dedicated global tables, for example all `*.page` files use a separate global table per URL request in NPL Web Server App.

Note : when loading an NPL file, we will first find if there is an up to date compiled version in the bin directory. if there is, we will load the compiled version, otherwise we will use the text version. use bin version, if source version does not exist; use bin version, if source and bin versions are both on disk (instead of zip) and that bin version is newer than the source version. e.g. we can compile source to bin directory with file extension ".o", e.g. "script/abc.lua" can be compiled to "bin/script/abc.o", The latter will be used if available and up-to-date.

Remark : one should be very careful when calling with bReload set to true, since this may lead to recursive reloading of the same file. If this occurs, it will generate C Stack overflow error message.

Parameters

- `filePath`: the local relative file path. If the file extension is ".dll", it will be treated as a plug-in. Examples: "NPLRouter.dll" load a C++ or C# dll. Please note that, in windows, it looks for NPLRonter.dll; in linux, it looks for ./libNPLRouter.so "plugin/libNPLRouter.dll" almost same as above, it is reformatted to remove the heading 'lib' when loading. In windows, it looks for plugin/NPLRonter.dll; in linux, it looks for ./plugin/libNPLRouter.so

- `bReload`: if true, the file will be reloaded even if it is already loaded. otherwise, the file will only be loaded if it is not loaded yet.

void **`load_`**(**const** char **filePath*, bool *bReload*)
for NPL managed only.

void **`load1`**(**const** object &*filePath*)
same as NPL.load(filepath, false);

void **`DoString`**(**const** object &*sCode*)
execute a given string immediately in protected mode.
Remark :caution: there may be a security issue.
Parameters

- `sCode`: : the code to run. the code can not be longer than some internally defined value.

void **`DoString2`**(**const** object &*sCode*, **const** char **sFilename*)
execute a given string immediately in protected mode.
Remark :caution: there may be a security issue.
Parameters

- `sCode`: : the code to run. the code can not be longer than some internally defined value.
- `sFilename`: can be nil, a filename to be associated with the chunk of code for debuggin purposes.

void **`DoString_`**(**const** char **sCode*)
only for NPL managed only

void **`test`**(**const** object &*input*)
this function should be removed in release build. it just run NPL C++ test cases
Parameters

- `input`: which test case to run.

bool **`SetTimer`**(**const** object &*nIDEvent*, float *fElapse*, **const** object &*sNeuronFile*)
creates a timer with the specified time-out value
Return : true if succeeds.An application can pass the value of the nIDEvent parameter to the NPL.KillTimer function to destroy the timer.
Parameters

- `nIDEvent`: Specifies a positive timer identifier. For nIDEvent<=0, they are reserved for internal uses. If the NPL runtime already has a timer with the value nIDEvent, then the existing timer is replaced by the new timer. When SetTimer replaces a timer, the timer is reset.
- `fElapse`: Specifies the time-out value, in seconds. Please note that a timer will not be repeatedly activated if its timeout is shorter than the frame rate of the NPL simulation pipeline .
- `sNeuronFile`: The NPL file to be activated when the time-out value elapses. it can also carry sCode. e.g. "script/hello.lua;funcABC();", ";funcABC();", "(gl)script/hello.lua";

bool **`KillTimer`**(**const** object &*nIDEvent*)
Destroys the specified timer
Return : If the function succeeds, the return value is true
Parameters

- `nIDEvent`: Specifies the timer to be destroyed.For nIDEvent<=0, they are reserved for internal uses can not be killed by this function. This value must be the same as the nIDEvent value passed to the SetTimer function that created the timer.

bool **`ChangeTimer`**(**const** object &*nIDEvent*, int *dueTime*, int *period*)

Changes the start time and the interval between method invocations for a timer, using 32-bit signed integers to measure time intervals.

Return : If the function succeeds, the return value is true

Parameters

- `nIDEvent`: Specifies the timer to be destroyed.For nIDEvent<=0, they are reserved for internal uses can not be killed by this function. This value must be the same as the nIDEvent value passed to the SetTimer function that created the timer.
- `dueTime`: The amount of time to delay before the invoking the callback method specified when the Timer was constructed, in milliseconds. Specify zero (0) to restart the timer immediately. however, the current implementation does not accept dueTime that is larger than MAX_TIMER_DUE_TIME 10000000, which is 10000 seconds.
- `period:The`: time interval between invocations of the callback method specified when the Timer was constructed, in milliseconds.

const string &**`SerializeToSCode`**(**const** char **sStorageVar*, **const** object &*input*)

serialize a luabind object into sCode. The object could be a value, string or a table of the above type. input also accepts *ParaFileObject* typed data. *ParaFileObject* will be converted to base64 string internally.There are size limit though of 10MB.

Return sCode the output scode

Parameters

- `sStorageVar`: if this is "", the scode contains only the object. otherwise an assignment is made, by prefixing the scode with "[sStorageVar = ".
- `input`: input luabind object

bool **`IsSCodePureData`**(**const** char **sCode*)

verify the script code. it returns true if the script code contains pure msg data or table. this function is used to verify scode received from the network. So that the execution of a pure data in the local runtime is harmless.

bool **`IsPureData`**(**const** char **sCode*)

it will return true if input string is "false", "true", NUMBER, STRING, and {table}

bool **`IsPureTable`**(**const** char **sCode*)

it will return true if input string is a {table} containing only "false", "true", NUMBER, STRING, and other such {table}

luabind::object **`LoadTableFromString`**(**const** object &*input*)

load a table from string.

luabind::object **`LoadObjectFromString`**(**const** object &*input*)

load a table,string, anything from string.

void **`StartNetServer`**(**const** object &*server*, **const** object &*port*)

start the NPL net server's io_service loop. This function returns immediately. it will spawn the accept and dispatcher thread. call this function only once per process.

Parameters

- `server`: if nil, it defaults to "127.0.0.1"
- `port`: if nil, it defaults to "60001"; if "0", we will not listen or accept incoming connections. This is usually the setting for pure client application.

void **`StopNetServer`**()

stop the net server

void **`AddPublicFile`**(**const** string &*filename*, int *nID*)

add a nID, filename pair to the public file list. we only allow remote NPL runtime to activate files in the public file list. Each public file has a user defined ID. The NPL network layer always try to use its ID for transmission to minimize bandwidth. There are some negotiations between sender and receiver to sync the string to ID map before they use it. [thread safe]

Parameters

- `nID`: the integer to encode the string. it is usually small and positive number.
- `sString`: the string for the id. if input is empty, it means removing the mapping of nID.

void **`RegisterEvent`** (int *nType*, **const** char **sID*, **const** char **sScript*)

register a network event handler

Parameters

- `nType`: reserved, this is always 0.
- `sID`: a string identifier of the network event handler.
- `sScript`: the script to be executed when the event is triggered.This is usually a function call in NPL. sScript should be in the following format "{NPL filename};{sCode};". this is the same format in the UI event handler. Please note, it is slightly faster when there is no sCode, and we can register the callback script to be in a different NPL state(thread), such as "(gateway)script/apps/GameServer/GSL_NPLEvent.lua".

void **`UnregisterEvent`** (int *nType*, **const** char **sID*)

unregister a network event handler

Parameters

- `nType`: reserved, this is always 0.

luabind::object **`GetStats`** (**const** object &*inout*)

get statistics about this runtime environment. e.g. local stats = NPL.GetStats({connection_count = true, nids_str=true, nids = true});

Return {connection_count = 10, nids_str="101,102,"}

Parameters

- `input`: this should be a table containing mapping from string to true. function will return a new table by replacing true with the actual data. such as {["connection_count"] = true, ["nids_str"] = true }, supported fields are "connection_count" : total connection. "nids_str": commar separated list of nids. "nids": a table array of nids

void **`ClearPublicFiles`** ()

clear all public files, so that the NPL server will be completely private. [thread safe]

bool **`AddNPLRuntimeAddress`** (**const** object &*npl_address*)

add a given NPL runtime address to the fully trusted server addresses list. whenever an address is added, one can activate any public files in that runtime. Please note that connection is only established on first activation. In most cases, one should add all trusted servers at start up time.

Return : true if successfully added.

Parameters

- `npl_address`: this should be a table of { host = "127.0.0.1", port = "60001", nid = "MyServer", } [thread safe]

string **`GetIP`** (**const** char **nid*)

get the ip address of given NPL connection. this function is usually called by the server for connected clients.

Return : the ip address in dot format. empty string is returned if connection can not be found.

Parameters

- `nid`: nid or tid.

void **`accept`** (**const** object &*tid*, **const** object &*nid*)

accept a given connection. The connection will be regarded as authenticated once accepted. [thread safe]

Parameters

- `tid`: the temporary id or NID of the connection to be accepted. usually it is from msg.tid or msg.nid.
- `nid`: if this is not nil, tid will be renamed to nid after accepted.

void **`accept_`**(**const** char **tid*, **const** char **nid*)
: this function is used by C++ API interface.

void **`reject`**(**const** object &*nid*)
: reject and close a given connection. The connection will be closed once rejected. [thread safe]

Parameters

- `nid`: the temporary id or NID of the connection to be rejected. usually it is from msg.tid or msg.nid. it can also be {nid=number|string, reason=0|1|-1} reason:
  - 0 or positive value is forcibly reset/disconnect (it will abort pending read/write immediately).
  - 1 is another user with same nid is authenticated. The server should send a message to tell the client about this.
  - -1 or negative value means gracefully close the connection when all pending data has been sent.

void **`reject_`**(**const** char **nid*, int *nReason* = 0)
: this function is used by C++ API interface.

void **`SetUseCompression`**(bool *bCompressIncoming*, bool *bCompressOutgoing*)
: whether to use compression on transport layer for incoming and outgoing connections

Parameters

- `bCompressIncoming`: if true, compression is used for all incoming connections. default to false.
- `bCompressIncoming`: if true, compression is used for all outgoing connections. default to false.

void **`SetCompressionKey`**(**const** object &*input*)
: set the compression method of incoming the outgoing messages. If this is not called, the default internal key is used for message encoding. [Not Thread Safe]: one must call this function before sending or receiving any encoded messages. so it is usually called when the game engine starts.

Parameters

- `input`: a table, such as { key = "", size = 100, UsePlainTextEncoding = 1}, or {UsePlainTextEncoding = 1} input.key: the byte array of key. the generic key that is used for encoding/decoding input.size: size in bytes of the sKey. default is 64 bytes input.UsePlainTextEncoding: default to 0. if 0, the key is used as it is. if 1, the input key will be modified so that the encoded message looks like plain text(this can be useful to pass some firewalls). if -1, the input key will be modified so that the encoded message is binary.

const char ***`GetSourceName`**()
: get the current activation's source name. Each NPL activation has a source name which started the activation. This is usually "" for local activation and some kind of "name@server" for network activation.

void **`SetSourceName`**(**const** char **sName*)
: Set the current activation's source name. Each NPL activation has a source name which started the activation. This function is called automatically when a new activation occurs.So only call this function if one wants to override the old one for special code logics.

Parameters

- `sName`: This is usually "" for local activation and some kind of "name@server" for network activation.

void **`EnableNetwork`**(bool *bEnable*, **const** char **CenterName*, **const** char **password*)
: Enable the network, by default the network layer is disabled. calling this function multiple time with different CenterName will restart the network layer with a different center name.

Return true if succeeded.

Parameters

- `bEnable`: true to enable, false to disable.If this is false, the CenterName and Password are ignored.
- `CenterName`: the local nerve center name. it is also the user name which local receptor will use in the credentials to login in other NPL runtime.
- `Password`:

void **`AddDNSRecord`**(**const** char **sDNSName*, **const** char **sAddress*)

add a DNS server record to the current NPL runtime. DNS server record is a mapping from name to (IP:port) if one maps several IP:port to the same name, the former ones will be overridden.

Parameters

- `sDNSName`: the DNS server name. the DNS name "_world" is used for the current world DNS server. It is commonly used as a DNS reference to the current world that the user is exploring.
- `sAddress`: "IP:port". e.g. "192.168.1.10:4000"

void **`SetDefaultChannel`**(int *channel_ID*)

Set the default channel ID, default value is 0. Default channel is used when NPL.activate() calls does not contain the channel property.

Parameters

- `channel_ID`: It can be a number in [0,15].default is 0

int **`GetDefaultChannel`**()

Get the default channel ID, default value is 0. Default channel is used when NPL.activate() calls does not contain the channel property.

Return channel_ID It can be a number in [0,15].default is 0

void **`SetChannelProperty`**(int *channel_ID*, int *priority*, int *reliability*)

Messages can be sent via predefined channels. There are 16 channels from 0 to 15 to be used. 0 is the default channel. This method sets the channel property for a given channel. The default channel property is given in table. The following table shows the default NPL channel properties. It is advised for users to stick to this default mapping when developing their own applications. Table 1. Default NPL channel properties channel_ID Priority Reliability Usage 0 med RELIABLE_ORDERED System message 1 med UNRELIABLE_SEQUENCED Character positions 2 med RELIABLE_ORDERED Large Simulation Object transmission, such as terrain height field. 4 med RELIABLE_ORDERED Chat message 14 med RELIABLE files transmission and advertisement 15 med RELIABLE_SEQUENCED Voice transmission 11-15 med RELIABLE_ORDERED

Parameters

- `channel_ID`:
- `priority`:
- `reliability`:

void **`ResetChannelProperties`**()

reset all 16 predefined channel properties. according to table1. Default NPL channel properties. see also NPL_SetChannelProperty The following table shows the default NPL channel properties. It is advised for users to stick to this default mapping when developing their own applications. Table 1. Default NPL channel properties channel_ID Priority Reliability Usage 0 med RELIABLE_ORDERED System message 1 med UNRELIABLE_SEQUENCED Character positions 2 med RELIABLE_ORDERED Large Simulation Object transmission, such as terrain height field. 4 med RELIABLE_ORDERED Chat message 14 med RELIABLE files transmission and advertisement 15 med RELIABLE_SEQUENCED Voice transmission 11-15 med RELIABLE_ORDERED

void **`GetChannelProperty`**(int *channel_ID*, int **priority*, int **reliability*)

see also NPL_SetChannelProperty

Parameters

- `channel_ID`:
- `priority`: [out]
- `reliability`: [out]

void **`RegisterWSCallBack`**(**const** char **sWebServiceFile*, **const** char **sCode*)

this method will associate a web service (method) with either a sCode, which will be called when the web service returned. The returned message, if any, will be passed via a global parameter called msg. If msg == nil, it always means that there is an error during processing, such as HTTP 404 not found. the error code either string or number will be returned in a global variable called msgerror. For example: function callbackFunc1() if(msg~=nil) then log(msg.username); error code in msgerror else log(tostring(msgerror)); error code in msgerror end end NPL.RegisterWSCallBack("http://paraengine.com/login.aspx",callbackFunc1); NPL.activate("http://paraengine.com/login.aspx", {username=lxz});

Parameters

- `sWebServiceFile`: URL of the web service
- `sCode`: code to be executed when the web service is called. When a two-way web service call is invoked; it internally will create a thread for the returning message. Please refer to .Net 3.0 network communication architecture.

void **`UnregisterWSCallBack`**(**const** char **sWebServiceFile*)

unregister web service call back for a given file.

Parameters

- `sWebServiceFile`: if this is nil or "", all call backs will be unregistered.

void **`AsyncDownload`**(**const** char **url*, **const** char **destFolder*, **const** char **callbackScript*, **const** char **DownloaderName*)

Asynchronously download a file or an HTTP web page from the url.

Parameters

- `destFolder:folder`: path or file path. if the destFolder contains a file extension, we will save the downloaded file as the destFolder otherwise, we will save the file as the combination of the destFolder and the filename returned from the remote target.
- `callbackScript`: script code to be called, a global variable called msg is assigned, as below if url is a file: msg = {DownloadState=""|"complete"|"terminated", totalFileSize=number, currentFileSize=number, PercentDone=number} if url is a web page: msg = {DownloadState=""|"complete"|"terminated", ContentType=string that contains "text/html", Headers=string of {name:value }, StatusCode=int, StatusDescription=string, ResponseUri=string of actual url that is responding. totalFileSize=number, currentFileSize=number, PercentDone=number}

void **`CancelDownload`**(**const** char **DownloaderName*)

cancel all asynchronous downloads that matches a certain downloader name pattern

Parameters

- `DownloaderName:regular`: expression. such as "proc1", "proc1.*", ".*"

int **`Download`**(**const** char **url*, **const** char **destFolder*, **const** char **callbackScript*, **const** char **DownloaderName*)

Synchronous call of the function *AsyncDownload()*. This function will not return until download is complete or an error occurs. this function is rarely used. *AsyncDownload()* is used.

Return :1 if succeed, 0 if fail

ParaScripting::*ParaNPLRuntimeState* **`CreateRuntimeState`**(**const** string *&name*, int *type_*)

create a new runtime state. this function is thread safe

Return the newly created state is returned. If an runtime state with the same non-empty name already exist, the old one is returned.

Parameters

- `name`: if "", it is an anonymous runtime state. otherwise it should be a unique name.

- `type_`: NPLRuntimeStateType, the runtime state type. enum NPLRuntimeStateType { / the default NPL runtime state, with all NPL and ParaEngine functions loaded. it will consume about 1MB memory at start up. NPLRuntimeStateType_NPL = 0, / the light-weighter NPL runtime state, with only NPL and very limited functions loaded. NPLRuntimeStateType_NPL_LITE, / it consumes nothing. and is usually used with DLL plugins. NPLRuntimeStateType_DLL, };

ParaScripting::*ParaNPLRuntimeState* **`GetRuntimeState`** (**const** string &*name*)
get a runtime state with an explicit name. this function is thread safe

ParaScripting::*ParaNPLRuntimeState* **`CreateGetRuntimeState`** (**const** string &*name*, int *type_*)
it get runtime state first, if none exist, it will create one and add it to the main threaded state

bool **`DeleteRuntimeState`** (*ParaNPLRuntimeState runtime_state*)
create a given runtime state. this function is thread safe

void **`Compile`** (**const** char **arguments*)
compile source The main advantages of precompiling chunks are: faster loading, protecting source code from accidental user changes, and off-line syntax checking. Precompiling does not imply faster execution because in npl chunks are always compiled into bytecodes before being executed. compiling simply allows those bytecodes to be saved in a file for later execution. compiling can produce a single output file containing the bytecodes for all source files given. By default, the output file is named luac.out, but you can change this with the -o option. e.g. NPL.Compile("-p -o bin/script/config.o script/config.lua");

Parameters

- `arguments`: "%s [options] [filenames]" "Available options are:\n" " - process stdin\n" " -l list\n" " -o name output to file " LUA_QL("name") " (default is \"s") " " -p parse only " " -s strip debug information " " -v show version information " " – stop handling options ",

bool **`AppendURLRequest1`** (**const** object &*url*, **const** char **sCallback*, **const** object &*sForm*, **const** char **sPoolName*)
Append URL request to a pool.

Parameters

- `pUrlTask`: must be new CURLRequestTask(), the ownership of the task is transfered to the manager. so the caller should never delete the pointer.
- `sPoolName`: the request pool name. If the pool does not exist, it will be created. If null, the default pool is used. Append URL request to a pool. HTTP Get: NPL.AppendURLRequest("paraengine.com", "callbackFunc()", {"name1", "value1", "name2", "value2", }, "r") HTTP Post: NPL.AppendURLRequest("paraengine.com", "callbackFunc()", {name1="value1", name2="value2"}, "r")
- `url`: usually a rest url.
- `sCallback`: a string callback function. it may begin with (runtime_state_name) such as "(main)my_function()", if no runtime state is provided, it is the main state(Not the calling thread). This prevents the user to use multiple threads to download to the same file location by mistake. the callback function to be called. a global msg={data, header, code, rcode} contains the result. the msg.data contains the response data, and msg.header contains the http header, and msg.code contains the return code from libcurl, msg.rcode contains HTTP/FTP status code(200 if succeed)
- `sForm`: if it contains name value pairs, HTTP POST is used, otherwise HTTP GET is used. note that if it contains an array of name, value, name, value, ..., they will be regarded as url parameters can inserted to url automatically. This applies regardless of whether http get or post is used. one can also post a file, like belows {name = {file="/tmp/test.txt", type="text/plain"}} {name = {file="dummy.html", data="<html><bold>bold</bold></html>, type="text/html"}} some predefined field name in sForm is request_timeout: milliseconds of request timeout e.g. {request_timeout=50000,}

- `sPoolName`: the request pool name. If the pool does not exist, it will be created. If null, the default pool is used. there are some reserved pool names used by ParaEngine. They are:
  - "d": download pool. default size is 2, for downloading files.
  - "r": rest pool. default size is 5, for REST like HTTP get/post calls.
  - "w": web pool. default size is 5, for web based requests.

string **`EncodeURLQuery`**(**const** char **baseUrl*, **const** object &*sParams*)

build a use query using base url plus additional query parameters. NPL.BuildURLQuery("paraengine.com", {"name1", "value1", "name2", "value2", }) [thread safe]

Return : result is like "paraengine.com?name1=value1&name2=value2", they already in encoded form.

Parameters

- `sParams;`: an array table, where odd index is name, even index is value.

std::string **`GetExternalIP`**()

get extern IP address of this computer.

bool **`ChangeRequestPoolSize`**(**const** char **sPoolName*, int *nCount*)

There is generally no limit to the number of requests sent. However, each pool has a specified maximum number of concurrent worker slots. the default number is 1. One can change this number with this function.

bool **`FromJson`**(**const** char **sJson*, **const** object &*output*)

convert json string to NPL object. Internally TinyJson is used.

Return true if succeed. false if parsing failed.

Parameters

- `sJson`: the json code to parse. the first level must be array or table. otherwise, false is returned.
- `output`: [in|out] it must be a table. and usually empty table. the output is written to this table.

const string &**`ToJson`**(**const** object &*output*)

convert from NPL/lua table object to json string. /uXXXX encoding is recognized in string.

bool **`Compress`**(**const** object &*output*)

compress using zlib/gzip, etc

Return return true if success.

Parameters

- `output`: {method="zlib|gzip", content=string, [level=number, windowBits=number,] result=string}

bool **`Decompress`**(**const** object &*output*)

compress using zlib/gzip, etc

Parameters

- `output`: {method="zlib|gzip", content=string, [level=number, windowBits=number,] result=string}

class `CParaEngine`

#include <ParaScriptingMisc.h> global game engine related functions, such as ParaEngineCore interface, copy right information, simple water marking

Public Static Functions

ParaAttributeObject **`GetAttributeObject`**()

get the attribute object of ParaEngine settings

void **GetAttributeObject_**(*ParaAttributeObject &output*)
for API exportation.

string **GetVersion**()
get ParaEngine version

ParaScripting::*ParaAttributeObject* **GetViewportAttributeObject**(int *nViewportIndex*)
get attribute object

bool **ForceRender**()
render the current frame and does not return until everything is presented to screen. this function is usually used to draw the animated loading screen.

bool **Sleep**(float *fSeconds*)
cause the main thread to sleep for the specified seconds.

bool **SaveParaXMesh**(**const** char **filename*, *ParaAssetObject &xmesh*, bool *bBinaryEncoding*)
save an existing mesh to file.
Return : return true if succeeds.
Parameters
- `filename`: file to be saved to. if this is “”, the xmesh entity’s file name will be used and appended with ”.x”
- `xmesh`: ParaX mesh object to export.
- `bBinaryEncoding`: true to use binary encoding.

ParaAssetObject **GetRenderTarget**()
return the current render target. calling this function a second time will make the returned object from the previous call invalid.

bool **SetRenderTarget**(*ParaAssetObject &pRenderTarget*)
set the render target of the object.

bool **StretchRect**(*ParaAssetObject &pSrcRenderTarget*, *ParaAssetObject &pDestRenderTarget*)
Copy render target content from one surface to another. they may be of different resolution

bool **DrawQuad**()
draw a full screen quad.
Note : one need to set the vertex declaration to be S0_POS_TEX0 prior to calling this function

bool **SetVertexDeclaration**(int *nIndex*)
Set declaration by id enum VERTEX_DECLARATION { S0_POS_TEX0, // all data in stream 0: position and tex0 S0_POS_NORM_TEX0, // all data in stream 0: position, normal and tex0 S0_POS_NORM_TEX0_INSTANCED, // all data in stream 0: position, normal and tex0, stream1:instanced data S0_POS_TEX0_COLOR, // all data in stream 0: position, tex0 and color S0_S1_S2_OCEAN_FFT, // for FFT ocean S0_S1_S2_S3_OCEAN_FFT, // for FFT ocean with terrain height field S0_POS_NORM_TEX0_TEX1, // all data in stream 0: position, normal tex0 and tex1 MAX_DECLARATIONS_NUM, };
See VERTEX_DECLARATION
Return : return true if successful.
Parameters
- `nIndex`: value is in

struct **FileSystemWatcher_NPLCallback**
callback to npl runtime

struct **lua_func_entry**
for registering functions

struct NPL_GetNidsArray_Iterator
Inherits from NPLConnectionCallBack

struct NPL_GetNidsStr_Iterator
Inherits from NPLConnectionCallBack

class ParaAsset
#include <ParaScriptingCommon.h> *ParaAsset* namespace contains a list of HAPI functions to manage resources(asset) used in game world composing, such as 3d models, textures, animations, sound, etc. Resources are used to create scene objects. assets of the same type must have different names. Assets must be initialized before they can be used, and this should be manually done in scripts by calling *Init()*.

Public Static Functions

bool **OpenArchive**(**const** char **strFileName*)
open the archive file(zip or pkg file) for further resource file searching. If any file can not be located on the disk file system, it will go on searching for it in the archive file. files in the archive file will be relative to the ParaEngine SDK root directory.
Parameters
- `strFileName`: the package file path name

bool **GeneratePkgFile**(**const** char **srcZip*, **const** char **destPkg*)
Generate a pkg file which is equivalent to the specified zip file. This function can only be called, when a zip file can be successfully loaded.
Return true if successful. It will overwrite existing file. Output file is at the save directory and filename but with pkg file extension.
Parameters
- `srcZip`: the zip file name from which to generate.
- `destPkg`: : destination file. if nil, it just rename the srcZip

bool **OpenArchive2**(**const** char **strFileName*, bool *bUseRelativePath*)
open the archive file(zip or pkg file) for further resource file searching. If any file can not be located on the disk file system, it will go on searching for it in the archive file.
Parameters
- `strFileName`: the package file path name
- `bUseRelativePath`: if this is true, files in the archive file will be relative to the parent directory of archive path.

bool **OpenArchiveEx**(**const** char **strFileName*, **const** char **sRootDir*)
add archive to manager
Parameters
- `strFileName`: path of the zip or pkg file.
- `sRootDir`: files in the archive will be regarded as relative to this this root directory. If this is "", there is no root directory set. such as "model/", "script/", characters after the last slash is always stripped off.

void **CloseArchive**(**const** string &*path*)
close an archive. When done with an archive, one should always close it. Too many opened archives will compromise the IO performances.

const char ***GetAssetServerUrl**()
when an asset is not found, we will try to find it via this url. e.g. if asset is "model/test.dds", and asset url is "http://asset.paraengine.com/", then we will fetch the asset via "http://asset.paraengine.com/model/test.dds" if the asset path is "", asset server will be disabled.

void **SetAssetServerUrl** (**const** char **path*)

when an asset is not found, we will try to find it via this url. e.g. if asset is "model/test.dds", and asset url is "http://asset.paraengine.com/", then we will fetch the asset via "http://asset.paraengine.com/model/test.dds" if the asset path is "", asset server will be disabled.

Parameters

- `path:if`: the asset path is "", asset server will be disabled.

void **GarbageCollect** ()

Garbage Collect all assets according to reference count. If the reference count is not maintained well by the user, this function is not effective as *UnloadAll()*.

See *UnloadAll()*.

void **Unload** (**const** char **assettype*)

Unload an asset by its type name. once an unloaded asset is used again, its device resource will be reinitialized.

See *UnloadAll()*

- "ParaX": *Unload* all ParaX models
- "StaticMesh": *Unload* all StaticMesh models
- "Texture": *Unload* all Texture
- "Sound": *Unload* all Sound
- "Font": *Unload* all Font TODO: currently only "*" is supported.

Parameters

- `strAssetName`: value and meaning are listed below
  - "*": Unload all assets.

void **UnloadAll** ()

unload all assets. once an unloaded asset is used again, its device resource will be reinitialized.

See *Unload()*

void **UnloadDatabase** ()

unload all databases.

void **Init** ()

initialize all objects that have not been created yet NOTE: always call this function at least once when finished creating a batch of assets assets created using any of the functions in this namespace can not be used until this function is called.

ParaScripting::*ParaAssetObject* **LoadEffectFile** (**const** char **strAssetName*, **const** char **strFilePath*)

Load effect file from a text or compiled HLSL file. It will return the old effect if effect is already loaded before.

ParaScripting::*ParaAssetObject* **GetEffectFile** (**const** char **strAssetName*)

load an effect file by its name. it will return an invalid effect if the effect is not found.

ParaAssetObject **LoadParaX** (**const** char **strAssetName*, **const** char **strFilePath*)

Load ParaX model asset, ParaX model file contains mesh, skeletal animation, etc. Currently ParaX and mdx file format is supported, please refer to ParaX file document for creating ParaX file based multianimation asset.

ParaScripting::*ParaAssetObject* **LoadDatabase** (**const** char **strAssetName*, **const** char **strFilePath*)

Load Database asset. it must be sqlite database at the moment.

ParaAssetObject **LoadStaticMesh** (**const** char **strAssetName*, **const** char **strFilePath*)

Load a X file based static mesh object. If any side of the mesh's bounding box is longer than 50 meters(units) and that the triangle count is over 1500, Octree will be used to sort its triangle lists, otherwise no culling optimization will be used when rendering the mesh. Static mesh is suitable for rendering small repeated object such as stones, trees, or large object such as level mesh. another usage is that it can be used as physical data to be fed to the physics engine, in which case simple and convex geometry should be used as much as possible.

ParaAssetObject **`LoadTexture`** (**const** char **strAssetName*, **const** char **strFilePath*, int *nSurfaceType*)

Note : we treat png file as DXT3 by default. if the texture filename ends with "_32bits.png", we will load with D3DFMT_A8R8G8B8 instead of DXT3. If one wants to ensure high resolution texture, use TGA format instead. All dds textures are loaded with full mipmapping default.

Return

Parameters

- `strAssetName`:
- `strFilePath`: if the file name ends with _a{0-9}{0-9}{0-9}.xxx, it will be regarded as a texture sequence. and the nSurfaceType will be ignored and forced to TextureSequence
- `nSurfaceType`: enum _SurfaceType { / render target, call SetTextureInfo() to specify size. if SetTextureInfo() / is never called, the back buffer size is used. RenderTarget = 0, / normal managed texture, with all mip-mapping level created StaticTexture = 1, / a group of textures, such as xxx_a001.jpg, ..., xxx_a009.jpg TextureSequence = 2, / texture in memory SysMemoryTexture, / BLP textures BlpTexture, / detailed terrain texture TerrainHighResTexture, / cube texture for environment mapping, etc. CubeTexture, }

ParaAssetObject **`LoadSpriteFrame`** (**const** char **strAssetName*, int *nFrames*, int *nRow*, int *nCol*)

A sprite object must be created from Sprite frame and a texture. This is to create the sprite frame, which tells the engine how the sprite is loaded in the texture.

Parameters

- `nFrames`: how many frames the sprite has
- `nRow`: number of rows in the texture
- `nCol`: number of columns in the texture

ParaAssetObject **`LoadFont`** (**const** char **strAssetName*, **const** char **FontName*, int *nFontSize*)

load a system font, such as arial, times new man, etc.

Parameters

- `nFontSize`: in pixels

ParaAssetObject **`LoadImageFont`** (**const** char **strAssetName*, **const** char **TextureName*, int *nSize*, int *nTxtColumns*)

load an image based font, not tested.

ParaAssetObject **`LoadSound`** (**const** char **strAssetName*, **const** char **strFilePath*, bool *bInit*)

load a sound or music. The sound is not initialized until it is played for the first time.

Parameters

- `bInit`: whether to initialize the file

void **`AddFontName`** (**const** string &*sLocalName*, **const** string &*sTypeFaceName*)

give an alias name to a given font name. The most common use of this function is to replace the "System" font with a custom game font installed at "fonts/" folder.

Parameters

- `sLocalName`: a local file name like "System", "Default"
- `sTypeFaceName`: the real type face name to use when creating the font. please note that, the engine will search for the file "fonts/[sTypeFaceName].ttf", if this file exists, it will use that it, instead of the system installed font if any. Note: game custom font files under "fonts/" must be named by their true font name (i.e. type face name), otherwise they will not be loaded properly.

int **`GetBoneAnimProviderIDByName`** (**const** char **sName*)

Get provider id by name. Name is used when creating the provider instance. It is usually the same as the file path from which animation data is loaded. return -1 if not found

const char ***`GetBoneAnimProviderFileNameByID`**(int *nAnimID*)

get the file name of a given animID. It may return NULL, if animID is invalid or is an internal animation id.

int **`CreateBoneAnimProvider`**(int *nAnimID*, **const** char **name*, **const** char **filename*, bool *bOverwrite*)

Create an animation provider from file.

Return : return the nAnimID, in case nAnim is set to -1. -1 is returned, if failed.

Parameters

- `nAnimID`: -1 if one wants it to be automatically assigned. otherwise one can manually specify one. Please note, if there is already a provider with the same ID, the old one is automatically released and replaced with the new one.
- `name`: optional key. Allowing user to query by a user friendly name. This can be NULL.
- `filename`: from which file the animation data is loaded. It can be a ParaX animation file or BVH file.
- `bOverwrite`: whether to overwrite existing with the same nAnimID or name

bool **`DeleteBoneAnimProvider`**(int *nAnimID*)

delete a provider by ID.

Return : return true if succeed.

int **`PrintToFile`**(**const** char **strFileName*, DWORD *dwSelection*)

print all asset file to a given file. Each asset is on a single line, in the following format: [AssetFile-Name]

Return : the number of results printed are returned.

Parameters

- `strFileName`: to which file the result is written to. if NULL or "", it is "temp/assets.txt"
- `dwSelection`: bitwise on which assets to export, 1 is for texture, 2 is for Mesh, 4 is for ParaXEntity. Default to 0xffffffff

bool **`Refresh`**(**const** char **filename*)

refresh asset if it is already loaded. it will search for all refreshable asset type, such as textures and mesh, etc. if found, it will call the *Refresh()* method on the asset entity and return true, or return false.

class `ParaAssetObject`

#include <ParaScriptingCommon.h> it represents an asset entity.

Public Functions

bool **`IsValid`**()

check if the object is valid

bool **`IsLoaded`**()

most assets are loaded asynchronously. This allows us to check if an asset is loaded. For example, we can *LoadAsset()* for a number of assets that need preloading. and then use a timer to check if they are initialized and remove from the uninialized list.

bool **`equals`**(**const** *ParaAssetObject obj*) **const**

whether the two objects are equal

bool **`Reload`**()

reload the asset from file. Please note that for scene nodes which are currently using the asset entities, they are not automatically updated. For example, the physics which depends on a mesh entity, will not be automatically updated, once the mesh entity is reloaded. This function is almost solely used for debugging.

Return : return true if the mesh is updated.

void **UnloadAsset**()
unload the asset from video and system memory. This is usually used for animated or one time texture entity. Please note, asset is unloaded, but can still be used by other scene objects.The use of an unloaded object will cause the object to be loaded again.

void **LoadAsset**()
Preload the asset to video and system memory, even though there is no scene object using the object in the previous frames.

void **Refresh**()
currently, this function only takes effects on texture entity refresh this entity with a local file.
Parameters
- `sFilename`: if NULL or empty, the old file will be used.

void **Release**()
call this function to safely release this asset. If there is no further reference to this object, it will actually delete itself (with "delete this"). So never keep a pointer to this class after you have released it. A macro like SAFE_RELEASE() is advised to be used.

int **GetRefCount**()
get the reference count

void **GarbageCollectMe**()
if its reference count is zero, unload this asset object. any reference holder of this object can call this function to free its resources, if they believe that it will not be needed for quite some time in future.

string **GetKeyName**()
get the key name. this is usually the file name of the entity. return "" if it is not valid.

const char ***GetKeyName_**()
this function shall never be called from the scripting interface. Solely used for managed exporting.

string **GetFileName**()
get the file name. this is always the file name of the entity. return "" if it is not valid.

const char ***GetFileName_**()
this function shall never be called from the scripting interface. Solely used for managed exporting.

int **GetType**()
get the asset type: enum AssetType { base=0, texture=1, mesh=2, multianimation=3, spritevertex, font, sound, mdx, parax, database, effectfile, dllplugin, };
Return -1 is returned, if the asset is invalid.

int **SetHandle**(int *nHandle*)
set an integer handle to this asset. This is usually used by effect file asset. We can later assign mesh's primary technique handler using this value. please note that handles are not automatically created for most custom asset, one needs to manually create one. call this function multiple times with different handle, will associate the same asset with multiple handles.
Return : handle of this asset after successful set.
Parameters
- `nHandle`: TODO: if nHandle is -1, the system will automatically allocate a free handle for it and returned.

int **GetHandle**()
Get the integer handle to this asset. if there are multiple handles, the first (smallest) handle is returned. if handle is not available. it will return -1 (INVALID handle).

ParaScripting::*ParaAttributeObject* **GetAttributeObject**()
get the attribute object associated with the current asset object, such as getting the poly count, etc

ParaScripting::*ParaParamBlock* **GetParamBlock**()
get the parameter block. currently only effect and mesh entity file asset has effect param block. Currently the effect parameters can be set via *ParaParamBlock* interface from the scripting interface. we offer three levels of effect parameters: per effect file, per asset file, per mesh object. Effect parameters are also applied in that order. e.g. per effect file affects all object rendering with the effect file; per asset file affects all objects that use the mesh asset file; per mesh object affects only the mesh object.

bool **Begin**()
only applies to effect entity: begin effect

bool **BeginPass**(int *nPass*)
only applies to effect entity: begin effect pass

void **SetTexture**(int *nIndex*, **const** char **filename*)
only used for effect file asset.
Parameters
- `nIndex`: texture stage
- `filename`: if "", it will set to NULL

bool **EndPass**()
only applies to effect entity: end effect pass

bool **End**()
only applies to effect entity: end effect

bool **CommitChanges**()
this apply to changes.

void **SetTextureFPS**(float *FPS*)
For animated textures. set the FPS for animation textures. this provides a short cut to animated textures
Parameters
- `nFPS`: frames per seconds. default value is 15 FPS

void **EnableTextureAutoAnimation**(bool *bEnable*)
For animated textures. whether to enable texture animation. this provides a short cut to animated textures
Parameters
- `bEnable`: default value is true. Set this to false, if one wants to manually animate the texture, such as controlling from the scripting interface.

void **SetCurrentFrameNumber**(int *nFrame*)
For animated textures. set the current frame number. this provides a short cut to animated textures
Parameters
- `nFrame`:

int **GetCurrentFrameNumber**()
For animated textures. Get the current frame number. this provides a short cut to animated textures
Return frame number is returned

int **GetFrameCount**()
For animated textures. Get the total frames in the animated texture. this provides a short cut to animated textures

Return frame number is returned

int **GetWidth**()
get the texture width
Return

int **GetHeight**()
get the texture height
Return

void **SetSize**(int *nWidth*, int *nHeight*)
set the texture info (size) of the asset

object **GetBoundingBox**(**const** object &*box*)
get the bounding box (AABB) of the mesh or parax entity in object space. This function returns nothing if asset is not mesh or character entity.
Parameters

- `box`: [inlout] a script table to receive the output. in the format: {min_x, min_y, min_z, max_x, max_y, max_z}

int **GetNumReplaceableTextures**()
get the total number of replaceable textures, which is the largest replaceable texture ID. but it does not mean that all ID contains valid replaceable textures. This function can be used to quickly decide whether the model contains replaceable textures. Generally we allow 32 replaceable textures per model.
Return 0 may be returned if no replaceable texture is used by the model.

ParaAssetObject **GetDefaultReplaceableTexture**(int *ReplaceableTextureID*)
get the default replaceable texture by its ID. The default replaceable texture is the main texture exported from the 3dsmax exporter.
Return this may return invalid asset, if replaceable texture is not set before or ID is invalid.
Parameters

- `ReplaceableTextureID`: usually [0-32) generally speaking, replaceable ID 0 is used for general purpose replaceable texture, ID 1 is for user defined. ID 2 is for custom skins.

Public Static Functions

TextureEntity ***GetTexture**(**const** object &*texture*)
static helper functions:
Parameters

- `texture`: it can be string or a *ParaAssetObject*

class **ParaAttributeObject**
#include <ParaScriptingGlobal.h> it represents an attribute object associated with an object. Call *ParaObject::GetAttributeObject()* or *ParaObject::GetAttributeObject()* to get an instance of this object. e.g. In NPL, one can write local att = player:*GetAttributeObject()*; local bGloble = att:GetField("global", true); local facing = att:GetField("facing", 0); att:SetField("facing", facing+3.14); local pos = att:GetField("position", {0,0,0}); pos[1] = pos[1]+100;pos[2] = 0;pos[3] = 10; att:SetField("position", pos); att:PrintObject("test.txt");

the following shows objects and their supported attributes.

Public Functions

ParaAttributeObject **GetAttributeObject**()
get the attribute object. This function return a clone of this object.

bool **equals**(**const** ParaAttributeObject &*obj*) **const**
return true, if this object is the same as the given object.

ParaScripting::*ParaAttributeObject* **GetChild**(**const** std::string &*sName*)
get child attribute object. this can be regarded as an intrusive data model of a given object. once you get an attribute object, you can use this model class to access all data in the hierarchy.

int **GetColumnCount**()
we support multi-dimensional child object. by default objects have only one column.

bool **AddChild**(*ParaAttributeObject* &*obj*)
add a child object

const *ParaObject* &**QueryObject**()
query object

bool **IsValid**() **const**
check if the object is valid

int **GetClassID**() **const**
class ID

const char ***GetClassName**() **const**
class name

const char ***GetClassDescription**() **const**
class description

void **SetOrder**(int *order*)
Set which order fields are saved. enum Field_Order { Sort_ByName, Sort_ByCategory, Sort_ByInstallOrder, };

int **GetOrder**()
get which order fields are saved.

int **GetFieldNum**()
get the total number of field.

const char ***GetFieldName**(int *nIndex*)
get field at the specified index. “” will be returned if index is out of range.

int **GetFieldIndex**(**const** char **sFieldname*)
get field index of a given field name. -1 will be returned if name not found.
Return
Parameters
- `sFieldname`:

const char ***GetFieldType**(int *nIndex*)
get the field type as string
Return one of the following type may be returned “void” “bool” “string” “int” “float” “float_float” “float_float_float” “double” “vector2” “vector3” “vector4” “enum” “deprecated” “”
Parameters
- `nIndex`: : index of the field

bool **IsFieldReadOnly**(int *nIndex*)
whether the field is read only. a field is ready only if and only if it has only a get method.
Return true if it is ready only or field does not exist
Parameters
- `nIndex`: : index of the field

const char ***GetFieldSchematics**(int *nIndex*)
Get Field Schematics string
Return “” will be returned if index is out of range
Parameters
- `nIndex`: index of the field

const char ***GetSchematicsType**(int *nIndex*)
parse the schema type from the schema string.
Return : simple schema type. it may be any of the following value. unspecified: “” color3 ”:rgb” file ”:file” script ”:script” integer ”:int”

void **GetSchematicsMinMax**(int *nIndex*, float *fMinIn*, float *fMaxIn*, float &*fMin*, float &*fMax*)
parse the schema min max value from the schema string.
Return true if found min max.
Parameters
- `nIndex`: index of the field
- `fMin`: : [in|out] default value
- `fMax`: : [in|out] default value

object **GetField**(**const** char **sFieldname*, **const** object &*output*)
get field by name. e.g. suppose att is the attribute object. local bGloble = att:GetField(“global”, true); local facing = att:GetField(“facing”, 0); local pos = att:GetField(“position”, {0,0,0}); pos[1] = pos[1]+100;pos[2] = 0;pos[3] = 10;

Return : return the field result. If field not found, output will be returned. if field type is vectorN, return a table with N items.Please note table index start from 1
Parameters
- `sFieldname`: field name
- `output`: default value. if field type is vectorN, output is a table with N items.

const char ***GetStringField**(**const** char **sFieldname*)
similar to *GetField()*. except that the output is a string. Used for API exporting. not thread safe.

double **GetValueField** (**const** char **sFieldname*, int *nIndex* = 0)
: similar to *GetField()*. except that the output is a value. Used for API exporting. not thread safe.

  Parameters
  - `nIndex`: if the value has multiple component, such as a vector3. this is the index of the componet.

void **SetField** (**const** char **sFieldname*, **const** object &*input*)
: set field by name e.g. suppose att is the attribute object. att:SetField("facing", 3.14); att:SetField("position", {100,0,0});

  Parameters
  - `sFieldname`: field name
  - `input`: input value. if field type is vectorN, input is a table with N items.

void **SetStringField** (**const** char **sFieldname*, **const** char **input*)
: similar to *SetField()*. except that the input is a string. Used for API exporting. not thread safe.

void **SetValueField** (**const** char **sFieldname*, int *nIndex*, double *value*)
: similar to *SetField()*. except that the input is a string. Used for API exporting. not thread safe.

  Parameters
  - `nIndex`: if the value has multiple component, such as a vector3. this is the index of the component.

void **CallField** (**const** char **sFieldname*)
: call field by name. This function is only valid when The field type is void. It simply calls the function associated with the field name.

void **PrintObject** (**const** char **file*)
: print attribute to file

  Parameters
  - `file`: file name to save the manual to.

bool **ResetField** (int *nFieldID*)
: Reset the field to its initial or default value.

  Return true if value is set; false if value not set.

  Parameters
  - `nFieldID`: : field ID

bool **InvokeEditor** (int *nFieldID*, **const** char **sParameters*)
: Invoke an (external) editor for a given field. This is usually for NPL script field

  Return true if editor is invoked, false if failed or field has no editor.

  Parameters
  - `nFieldID`: : field ID
  - `sParameters`: : the parameter passed to the editor

object **GetDynamicField** (**const** char **sFieldname*, **const** object &*output*)
: get field by name. e.g. suppose att is the attribute object. local bGloble = att:GetField("URL", nil); local facing = att:GetField("Title", "default one");

  Return : return the field result. If field not found, output will be returned. if field type is vectorN, return a table with N items.Please note table index start from 1

  Parameters
  - `sFieldname`: field name
  - `output`: default value. if field type is vectorN, output is a table with N items.

object **GetDynamicField_** (int *nIndex*, **const** object &*output*)
: Get a dynamic field with a given index.

const char ***GetDynamicFieldNameByIndex**(int *nIndex*)
get field name by index

int **GetDynamicFieldCount**()
how many dynamic field this object currently have.

int **SetDynamicField**(**const** char **sFieldname*, **const** object &*input*)
set field by name e.g. suppose att is the attribute object. att:SetDynamicField("URL", 3.14); att:SetDynamicField("Title", {100,0,0});
Return : -1 failed, if 0 means modified, if 1 means a new key is added, if 2 means a key is removed.
Parameters

- `sFieldname`: field name
- `input`: input value. can be value or string type

void **RemoveAllDynamicFields**()
remove all dynamic fields

int **AddDynamicField**(**const** std::string &*sName*, int *dwType*)
add dynamic field and return field index
Parameters

- `dwType`: type of ATTRIBUTE_FIELDTYPE

class ParaAudio
#include <ParaScriptingAudio.h> Audio Engine functions

Public Static Functions

bool **IsAudioEngineEnabled**()
————- Audio Engine Functions ————- get is audio engine enabled

void **EnableAudioEngine**(bool *bEnable*)
enable Audio Engine

void **SetVolume**(float *fVolume*)
Set the volume of all categories and all currently playing wave files.
Parameters

- `fVolume`: usually between [0,1], where 0 is silent and 1 is full. value larger than 1 is also possible.

float **GetVolume**()
Get the volume of average if all categories
Return usually between [0,1], where 0 is silent and 1 is full. value larger than 1 is also possible.

ParaAudioSource **Create**(**const** char **sName*, **const** char **sWavePath*, bool *bStream*)
create a given audio source by name. If no audio source with the name is loaded before, we will create one new; otherwise we will overwrite the previous one.
Return CAudioSource2_ptr object returned. It may be null if failed.
Parameters

- `sName`: the audio source name. Usually same as the audio file path, however it can be any string.
- `sWavePath`: if NULL, it will defaults to sName. Please note, in order to play the same music at the same time, they must be created with different names.
- `bStream`: whether to stream the music once created.

ParaAudioSource **Get**(**const** char **sName*)
get audio source by name. The source should be created by *Create()* function.

ParaAudioSource **`CreateGet`** (**const** char **sName*, **const** char **sWavePath*, bool *bStream*)

get a given audio source by name. If no audio source with the name is loaded before, we will create one.

Return CAudioSource2_ptr object returned. It may be null if failed.

Parameters

- `sName`: the audio source name. Usually same as the audio file path, however it can be any string.
- `sWavePath`: if NULL, it will defaults to sName. Please note, in order to play the same music at the same time, they must be created with different names.
- `bStream`: whether to stream the music once created.

void **`SetDistanceModel`** (int *eDistModel*)

set the audio distance model. see: http://connect.creativelabs.com/openal/Documentation/OpenAL%201.1%20Specification.htm

Parameters

- `eDistModel`: int of following. enum ParaAudioDistanceModelEnum { Audio_DistModel_NONE = 0, Audio_DistModel_INVERSE_DISTANCE, Audio_DistModel_INVERSE_DISTANCE_CLAMPED, Audio_DistModel_LINEAR_DISTANCE, Audio_DistModel_LINEAR_DISTANCE_CLAMPED, Audio_DistModel_EXPONENT_DISTANCE, Audio_DistModel_EXPONENT_DISTANCE_CLAMPED, };

bool **`PlayWaveFile`** (**const** char **szWavePath*, int *nLoop*)

Prepare and play a wave object from a standard wave file (wav, mp3, ogg/vorbis). If a wave file is already prepared before. It will be reused.

Parameters

- `szWavePath`: Path to the wave file. It can be from asset_manifest or relative to current directory path.
- `nLoop`: 0 means non-looping. 1 means looping.

int **`PlayMidiMsg`** (DWORD *dwMsg*)

more information, please see: midiOutShortMsg

Parameters

- `dwMsg`: MIDI message. The message is packed into a DWORD value with the first byte of the message in the low-order byte. The message is packed into this parameter as follows.

bool **`StopWaveFile`** (**const** char **szWavePath*, bool *bImmediateStop*)

stop a wave file

Parameters

- `szWavePath`: Path to the wave file.
- `bImmediateStop`: if false, it plays the wave to completion, then stops. For looping waves, this flag plays the current iteration to completion, then stops (ignoring any subsequent iterations). In either case, any release (or tail) is played. To stop the wave immediately, use true.

bool **`ReleaseWaveFile`** (**const** char **szWavePath*)

release a wave file

Parameters

- `szWavePath`: Path to the wave file.

class `ParaAudioSource`

#include <ParaScriptingAudio.h> It represents a 2D or 3D audio source object.

Public Functions

bool **`IsValid`**`()` **const**
: true if valid

void **`release`**`()`
: stop and unload this audio source from memory. It will make the sound source invalid after calling this function. it is good practice to unload unused sound.

bool **`play`**`()`
: Plays the source with the last set parameters.
: **Return** True if the source is playing, false if not.

const char ***`GetName`**`()`
: get the source name. (this may not be the file name)

bool **`play2d`**`(`bool *toLoop*`)`
: Plays the source in 2D mode. No automatic attenuation or panning will take place in this mode, but using setPosition will allow you to manually pan mono audio streams.
: **Return** True if the source is playing, false if not.
: **Parameters**
  - `toLoop`: Whether to loop (restart) the audio when the end is reached.

bool **`play3d`**`(`float *x*, float *y*, float *z*, float *soundstr*, bool *toLoop*`)`
: Plays the source in 3D mode.
: **Return** True if the source is playing, false if not.
: **Parameters**
  - `position`: Position to start the sound off at.
  - `soundstr`: Affects how the source attenuates due to distance. Higher values cause the source to stand out more over distance.
  - `toLoop`: Whether to loop (restart) the audio when the end is reached.

void **`pause`**`()`
: Pauses playback of the sound source.

void **`stop`**`()`
: Stops playback of the sound source.

void **`loop`**`(`bool *toLoop*`)`
: Controls whether the source should loop or not.
: **Parameters**
  - `toLoop`: Whether to loop (restart) the audio when the end is reached.

bool **`seek`**`(`float *seconds*, bool *relative*`)`
: Seeks through the audio stream to a specific spot. Note: May not be supported by all codecs.
: **Return** True on success, False if the codec does not support seeking.
: **Parameters**
  - `seconds`: Number of seconds to seek.
  - `relative`: Whether to seek from the current position or the start of the stream.

float **`getTotalAudioTime`**`()`
: **Return** the total amount of time in the audio stream. See IAudioDecoder for details.

int **`getTotalAudioSize`**`()`
: **Return** the total decoded size of the audio stream. See IAudioDecoder for details.

float **`getCurrentAudioTime`**()

Return the current position in the audio stream in seconds. See IAudioDecoder for details.

int **`getCurrentAudioPosition`**()

Return the current position in the decoded audio stream in bytes. See IAudioDecoder for details.

bool **`isValid`**() **const**

Return if the source is ready to be used.

bool **`isPlaying`**() **const**

Return if the source is playing.

bool **`isPaused`**() **const**

Return if the source is paused.

bool **`isStopped`**() **const**

Return if the source is stopped.

bool **`isLooping`**() **const**

Return if the source is looping.

void **`setPosition`**(float *x*, float *y*, float *z*)

Sets the position of the source in 3D space.

Parameters

- `position`: A 3D vector giving the new location to put this source.

void **`setVelocity`**(float *x*, float *y*, float *z*)

Sets the current velocity of the source for doppler effects.

Parameters

- `velocity`: A 3D vector giving the speed and direction that the source is moving.

void **`setDirection`**(float *x*, float *y*, float *z*)

Sets the direction the source is facing.

Parameters

- `direction`: A 3D vector giving the direction that the source is aiming.

void **`setRolloffFactor`**(float *rolloff*)

Sets the factor used in attenuating the source over distance. Larger values make it attenuate faster, smaller values make the source carry better. Range: 0.0f to +inf (Default: 1.0f).

Parameters

- `rolloff`: The rolloff factor to apply to the attenuation calculation.

void **`setStrength`**(float *soundstrength*)

Sets how well the source carries over distance.

Same as setRolloffFactor(1.0f/soundstrength). Range: 0.0f to +inf (Default: 1.0f).

Parameters

- `soundstrength`: How well the sound carries over distance.

void **`setMinDistance`**(float *minDistance*)

Sets the distance from the source where attenuation will begin. Range: 0.0f to +inf

Parameters

- `minDistance`: Distance from the source where attenuation begins.

void **`setMaxDistance`**(float *maxDistance*)

Sets the distance from the source where attenuation will stop. Range: 0.0f to +inf

Parameters

- `maxDistance`: Distance where attenuation will cease. Normally the farthest range you can hear the source.

void **`setPitch`** (float *pitch*)
Sets the pitch of the source. Range: 0.0f to +inf (Default: 1.0f)
Parameters
- `pitch`: New pitch level. Note that higher values will speed up the playback of the sound.

void **`setVolume`** (float *volume*)
Sets the source volume before attenuation and other effects. Range: 0.0f to +inf (Default: 1.0f).
Parameters
- `volume`: New volume of the source.

void **`setMinVolume`** (float *minVolume*)
Sets the minimum volume that the source can be attenuated to. Range: 0.0f to +inf (Default: 0.0f).
Parameters
- `minVolume`: New minimum volume of the source.

void **`setMaxVolume`** (float *maxVolume*)
Sets the maximum volume that the source can achieve. Range: 0.0f to +inf (Default: 1.0f).
Parameters
- `maxVolume`: New maximum volume of the source.

void **`setInnerConeAngle`** (float *innerAngle*)
Sets the angle of the inner sound cone of the source. The cone opens up in the direction of the source as set by *setDirection()*. Note: This causes the sound to be loudest only if the listener is inside this cone. Range: 0.0f to 360.0f (Default: 360.0f).
Parameters
- `innerAngle`: Inside angle of the cone.

void **`setOuterConeAngle`** (float *outerAngle*)
Sets the angle of the outer sound cone of the source. The cone opens up in the direction of the source as set by *setDirection()*. Note: If the listener is outside of this cone, the sound cannot be heard. Between the inner cone angle and this angle, the sound volume will fall off. Range: 0.0f to 360.0f (Default: 360.0f).
Parameters
- `outerAngle`: Outside angle of the cone.

void **`setOuterConeVolume`** (float *outerVolume*)
Sets how much the volume of the source is scaled in the outer cone. Range: 0.0f to +inf (Default: 0.0f).
Parameters
- `outerVolume`: Volume of the source in the outside cone.

void **`move`** (float *x*, float *y*, float *z*)
Convenience function to automatically set the velocity and position for you in a single call. Velocity will be set to new position - last position.
Parameters
- `position`: Position to move the source to.

void **`getPosition`** (float &*x*, float &*y*, float &*z*) **const**
Return the audio objects position

void **`getVelocity`** (float &*x*, float &*y*, float &*z*) **const**
Return the audio objects velocity

void **`getDirection`** (float &*x*, float &*y*, float &*z*) **const**

Return the audio objects direction

float **getRolloffFactor**() **const**
Return the factor used in attenuating the source over distance

float **getStrength**() **const**
Return the strength of the source

float **getMinDistance**() **const**
Return the distance from the source where attenuation will begin

float **getMaxDistance**() **const**
Return the distance from the source where attenuation will stop

float **getPitch**() **const**
Return the pitch of the source

float **getVolume**() **const**
Return the source volume before attenuation and other effects

float **getMinVolume**() **const**
Return the minimum volume that the source can be attenuated to

float **getMaxVolume**() **const**
Return the maximum volume that the source can achieve

float **getInnerConeAngle**() **const**
Return the angle of the inner sound cone of the source

float **getOuterConeAngle**() **const**
Return the angle of the outer sound cone of the source

float **getOuterConeVolume**() **const**
Return how much the volume of the source is scaled in the outer cone

class ParaBlockWorld
#include <ParaScriptingBlockWorld.h> Wrapper of internal CBlockWorld. may have multiple instance of the block world.

Public Static Functions

luabind::object **GetWorld**(**const** object &*sWorldName*)
static function to create get a world instance

int **GetVersion**(**const** object &*pWorld*)
get version

ParaScripting::*ParaAttributeObject* **GetBlockAttributeObject**(**const** object &*pWorld*)
get block terrain manager's attribute object.

bool **RegisterBlockTemplate**(**const** object &*pWorld*, uint16_t *templateId*, **const** object &*params*)
register blocks with given parameters
Parameters

- `params`: it can be attFlag of int type. or it can be a table containing additional format such as {attFlag=number, modelName=string, etc. }

void **SetBlockWorldYOffset** (**const** object *&pWorld*, float *offset*)
set Block world's y offset in real world coordinate.

void **EnterWorld** (**const** object *&pWorld*, **const** char **sWorldDir*)
call this function after all block templates has been registered to initialize the world note this function can be called multiple times to load different world with the same block templates. call *LeaveWorld()* before EnterWorld again.
Parameters
- `sWorldDir`: world directory or world config file.

void **LeaveWorld** (**const** object *&pWorld*)
call this function when leave the block world

void **LoadRegion** (**const** object *&pWorld*, uint16_t *x*, uint16_t *y*, uint16_t *z*)
load region at the given position. current implementation will load entire region rather than just chunk. one need to call load chunk before SetBlock/GetBlock api can be called in the region.

void **UnloadRegion** (**const** object *&pWorld*, uint16_t *x*, uint16_t *y*, uint16_t *z*)
unload data for a given region from memory

void **SetBlockId** (**const** object *&pWorld*, uint16_t *x*, uint16_t *y*, uint16_t *z*, uint32_t *templateId*)
set block id set the given position.
Parameters
- `xyz`: should be positive value
- `templateId`: template id. specify 0 to delete a block.

uint32_t **GetBlockId** (**const** object *&pWorld*, uint16_t *x*, uint16_t *y*, uint16_t *z*)
get block id at the given block position.

void **SetBlockData** (**const** object *&pWorld*, uint16_t *x*, uint16_t *y*, uint16_t *z*, uint32_t *data*)
set per block user data

uint32_t **GetBlockData** (**const** object *&pWorld*, uint16_t *x*, uint16_t *y*, uint16_t *z*)
get per block user data

luabind::object **GetBlocksInRegion** (**const** object *&pWorld*, int32_t *startChunkX*, int32_t *startChunkY*, int32_t *startChunkZ*, int32_t *endChunkX*, int32_t *endChunkY*, int32_t *endChunkZ*, uint32_t *matchType*, **const** object *&result*)
get block in [startChunk,endChunk]
Return {count,x{},y{},z{},tempId{}}
Parameters
- `result`: in/out containing the result.
- `startChunkYendChunkY`: if negative, and startChunkY == endChunkY, -startChunkY will be used as verticalSectionFilter (a bitwise filter).

void **SetBlockWorldSunIntensity** (**const** object *&pWorld*, float *value*)
set current sun intensity in [0,1] range

int **FindFirstBlock** (**const** object *&pWorld*, uint16_t *x*, uint16_t *y*, uint16_t *z*, uint16_t *nSide* = 4, uint32_t *max_dist* = 32, uint32_t *attrFilter* = 0xffffffff, int *nCategoryID* = -1)
find a block in the side direction that matched filter from block(x,y,z) this function can be used to check for free space upward or download
Return -1 if not found. otherwise distance to the first block that match in the side direction is returned.
Parameters

- `side`: 4 is top. 5 is bottom.
- `attrFilter`: attribute to match. 0 means air. default to any block
- `nCategoryID`: -1 means any category_id. default to -1

int **`GetFirstBlock`** (**const** object *&pWorld*, uint16_t *x*, uint16_t *y*, uint16_t *z*, int *nBlockId*, uint16_t *nSide* = 4, uint32_t *max_dist* = 32)

get the y pos of the first block of nBlockID, start searching from x, y, z in the side direction

void **`SetTemplateTexture`** (**const** object *&pWorld*, uint16_t *templateId*, **const** char **fileName*)

set the template texture. only used on client side This function is deprecated. use RegisterBlockTemplate instead.

luabind::object **`GetVisibleChunkRegion`** (**const** object *&pWorld*, **const** object *&result*)

get visible chunk region only used on client side

Return : world space chunk id {minX,minY,minZ,maxX,maxY,maxZ}

luabind::object **`Pick`** (**const** object *&pWorld*, float *rayX*, float *rayY*, float *rayZ*, float *dirX*, float *dirY*, float *dirZ*, float *fMaxDistance*, **const** object *&result*, uint32_t *filter* = 0xffffffff)

ray origin should be positive value, ray direction should be normalized value function is only used on client world

Return : result["x"] = pickResult.X; result["y"] = pickResult.Y; result["z"] = pickResult.Z; result["blockX"] = pickResult.BlockX; result["blockY"] = pickResult.BlockY; result["blockZ"] = pickResult.BlockZ; result["side"] = pickResult.Side; result["length"] = pickResult.Distance; side value : 0 negativeX,1 positiveX,2 NZ,3 PZ,4 NY, 5PY length > fMaxDistance when no collision detected

luabind::object **`MousePick`** (**const** object *&pWorld*, float *fMaxDistance*, **const** object *&result*, uint32_t *filter* = 0xffffffff)

picking by current mouse position. only used on client world

void **`SelectBlock`** (**const** object *&pWorld*, uint16_t *x*, uint16_t *y*, uint16_t *z*, bool *isSelect*)

add/remove block to/from highlight block list only used on client side

Parameters

- `xyz`: world space block id;
- `isSelect`: : true to select block, false to de-select block
- `nGroupID`: group id. 0 for highlight 1 for wireframe.

void **`DeselectAllBlock1`** (**const** object *&pWorld*, int *nGroupID*)

Parameters

- `nGroupID`: 0 for animated selection, 1 for wire frame selection. -1 for all

void **`SetDamagedBlock`** (**const** object *&pWorld*, uint16_t *x*, uint16_t *y*, uint16_t *z*)

set damage block id only used on client side

Parameters

- `xyz`: :world space block id;

void **`SetDamageDegree`** (**const** object *&pWorld*, float *damagedDegree*)

set damage block degree

Parameters

- `damageDegree`: [0,1] 0 means undamaged block,1 full damaged block

class **`ParaBootStrapper`**

#include <ParaScriptingGlobal.h> ParaGlobal namespace contains a list of HAPI functions to access the boot strapper functionality.

Bootstrapper file is a xml file to be executed at the very beginning of the game engine. Its main function is to specify the main game loop file to be activated. When the ParaIDE set a solution to be active, it actually modifies the bootstrapper file to load the main file of that application solution.

Public Static Functions

bool **`LoadFromFile`**(**const** char **sXMLfile*)
load from a given XML file.
Return true if success
Parameters
- `sXMLfile`: the path of the file, if this is "", the config/bootstrapper.xml will be used.

bool **`SaveToFile`**(**const** char **sXMLfile*)
save to a given XML file.
Return true if success
Parameters
- `sXMLfile`: the path of the file, if this is "", the config/bootstrapper.xml will be used.

void **`LoadDefault`**()
load the default setting. this function is called at the constructor.

const char ***`GetMainLoopFile`**()
get the game loop file. the game loop file provides the heart beat of the application. It is also the very first(entry) script to be activated when the application starts up. The default game loop is ./script/gameinterface.lua

void **`SetMainLoopFile`**(**const** char **sMainFile*)
Set the game loop file. the game loop file provides the heart beat of the application. It is also the very first(entry) script to be activated when the application starts up. The default game loop is ./script/gameinterface.lua

class **`ParaBrowserManager`**
#include <ParaScriptingBrowserManager.h> managing HTML browsers

Public Static Functions

static *ParaHTMLBrowser* **`GetBrowserWindow`**(**const** char **sFileName*)
get *ParaHTMLBrowser* by name. this function does not create any player if there is no browser with the given name.

static *ParaHTMLBrowser* **`GetBrowserWindow1`**(int *nWindowID*)
get *ParaHTMLBrowser* by nWindowID. this function does not create any player if there is no browser at the given index.

static *ParaHTMLBrowser* **`createBrowserWindow`**(**const** char **sFileName*, int *browserWindowWidth*, int *browserWindowHeight*)
create a new browser window with the given name and dimension in pixels.

static void **`onPageChanged`**(**const** object &*strScriptName*)
set event handler. the scripting interface will receive a msg table for the following type msg={windowid=number, value=[ValueInt|ValueString|nil]} where value is nil
Parameters
- `strScriptName`: format is "[neuronfile];sCode";

static void **`onNavigateBegin`**(**const** object &*strScriptName*)
set event handler. the scripting interface will receive a msg table for the following type msg={windowid=number, value=[ValueInt|ValueString|nil]} where value is a string
Parameters
- `strScriptName`: format is "[neuronfile];sCode";

static void **`onNavigateComplete`** (**const** object &*strScriptName*)
set event handler. the scripting interface will receive a msg table for the following type msg={windowid=number, value=[ValueInt|ValueString|nil]} where value is a string
Parameters
- `strScriptName`: format is “[neuronfile];sCode”;

static void **`onUpdateProgress`** (**const** object &*strScriptName*)
set event handler. the scripting interface will receive a msg table for the following type msg={windowid=number, value=[ValueInt|ValueString|nil]} where value is a int of [0-100]
Parameters
- `strScriptName`: format is “[neuronfile];sCode”;

static void **`onStatusTextChange`** (**const** object &*strScriptName*)
set event handler. the scripting interface will receive a msg table for the following type msg={windowid=number, value=[ValueInt|ValueString|nil]} where value is a string of status text
Parameters
- `strScriptName`: format is “[neuronfile];sCode”;

static void **`onLocationChange`** (**const** object &*strScriptName*)
set event handler. the scripting interface will receive a msg table for the following type msg={windowid=number, value=[ValueInt|ValueString|nil]} where value is a string
Parameters
- `strScriptName`: format is “[neuronfile];sCode”;

static void **`onClickLinkHref`** (**const** object &*strScriptName*)
set event handler. the scripting interface will receive a msg table for the following type msg={windowid=number, value=[ValueInt|ValueString|nil]} where value is a string of HRef
Parameters
- `strScriptName`: format is “[neuronfile];sCode”;

class **`ParaCamera`**
#include <ParaScriptingScene.h> The camera controller. First call *FollowObject()* method to set the character to follow. set call any of the camera transition functions to set the camera follow mode

Public Static Functions

void **`FollowObject`** (*ParaObject obj*)
set the character to follow. the object is also set as the current player
Parameters
- `obj`: the *ParaObject* to follow, one can call

void **`FollowObject`** (**const** char **strObjectName*)
set the character to follow. the object must be global in order to be found by its name. It is also set as the current player currently,
Parameters
- `strObjectName`: the name of the object to follow

void **`FirstPerson`** (int *nTransitionMode*, float *fHeight*, float *fAngle*)
Set the current camera to follow a certain object, using first person view of the object.
Parameters
- `nTransitionMode`: 1 (default) smooth move or 0 immediate move
- `fRadius`: the distance from the person to the camera
- `fAngle`: camera lift up angle, this is usually Pi/4: 45 degrees

void **`ThirdPerson`** (int *nTransitionMode*, float *fHeight*, float *fFacing*, float *fAngle*)
Set the current camera to follow a certain object, using Third Person view of the object, where character is always centered while allowing rotation around it from 360 degree angles,
Parameters
- `nTransitionMode`: 1 (default) smooth move or 0 immediate move
- `fRadius`: the distance from the person to the camera
- `fFacing`: camera facing, around the y axis, which is the world height axis.
- `fAngle`: camera lift up angle, this is usually Pi/4: 45 degrees

void **`Default`** (int *nTransitionMode*, float *fHeight*, float *fAngle*)
: Set the current camera to follow a certain object, using the default view of the object. character restricted to move within a rectangular region, while camera is facing a fixed direction.
Parameters
- `nTransitionMode`: 1 (default) smooth move or 0 immediate move
- `fHeight`: relative camera height above the object.
- `fAngle`: camera lift up angle, this is usually Pi/4: 45 degrees

void **`Fixed`** (int *nTransitionMode*, float *fX*, float *fY*, float *fZ*)
Set the current camera to look at a certain object, from a fixed camera location while allowing rotation around it from 360 degree angles,
Parameters
- `nTransitionMode`: 1 (default) smooth move or 0 immediate move
- `fX`: camera location: x
- `fY`: camera location: y
- `fZ`: camera location: z

void **`GetPosition`** (float **x*, float **y*, float **z*)
get the world position of the camera eye. This function takes no parameters. x,y,z are not input, but pure output. In the script, we can call it as below x,y,z = *ParaCamera.GetPosition()*; get the camera's eye position in Luabind, it is defined as .def("GetPosition", &*ParaCamera::GetPosition*, pure_out_value(_1) + pure_out_value(_2) + pure_out_value(_3)) please note, y is the absolute position in world coordinate
See SetPosition(float x, float y, float z)

void **`GetLookAtPosition`** (float **x*, float **y*, float **z*)
get the position that the camera is currently looking at.

void **`SetKeyMap`** (int *key*, int *scancode*)
we can alter key map at runtime
Parameters
- `key`: CharacterAndCameraKeys . [0-10]
- `scancode`: DIK_A, DIK_D,DIK_W,DIK_S,DIK_Q,DIK_E,DIK_SPACE,0,0,DIK_INSERT,DIK_DELETE

int **`GetKeyMap`** (int *key*)
get scancode from key id

ParaAttributeObject **`GetAttributeObject`** ()
get the attribute object associated with the current camera object.

void **`GetAttributeObject_`** (*ParaAttributeObject* &*output*)
used for API exportation.

class **`ParaCharacter`**
#include <ParaScriptingCharacter.h> *ParaObject* class: it is used to control game scene objects from scripts.

Public Functions

bool **IsValid**()
check if the object is valid

void **SetSpeedScale**(float *fSpeedScale*)
the default value of speed scale is 1.0f, which will use the character's default speed.

void **FallDown**()
if the biped is in air, it will fall down. In case a biped is put to stop and the terrain below it changes. one should manually call this function to let the biped fall down. Internally it just set the vertical speed to a small value

void **SetSizeScale**(float *fScale*)
the default value of size scale is 1.0f, which will use the character's default size. increasing this value will enlarge the character as well as its physical radius.

void **SetFocus**()
ask the camera to follow this character.The camera will be immediately focused on this character without translation.

void **AddAction2**(int *nAction*, **const** object &*param*)
add an action symbol, and let the character state manager determine which states it should go to. this function will not perform any concrete actions on the biped objects.

Parameters

- `nAction`: please see script/ide/action_table.lua for a list of valid values.
- `param`: the param specifying more details about the action. This value default to nil if nAct is S_ACTIONKEY, then this is const ActionKey* if nAct is S_WALK_POINT, then this is nil or 1, specifying whether to use angle.

bool **WalkingOrRunning**()
return true if character uses walk as the default moving animation.otherwise it uses running.

bool **HasAnimation**(int *nAnimID*)
whether an animation id exist. this function may have different return value when asset is async loaded.

Parameters

- `nAnimID`: predefined id.

bool **IsCustomModel**()
check whether this character is customizable.

void **ResetBaseModel**(*ParaAssetObject assetCharBaseModel*)
Reset base model. Base model is the frame model, to which other sub-models like weapons and clothing are attached. NOTE: the current version will delete the entire model instance, so that any sub-models attached to this model will be deleted and that the default appearance of the base model will show up, if one does not update its equipment after this call.

Parameters

- `assetCharBaseModel`: It is the new base model asset;it should be a valid ParaX model asset.

void **SetSkin**(int *nIndex*)
set the replaceable skin according to the skin database. this only applies to non-customizable characters.if the index exceeds, it will use the default one. the default skin is at index 0.

Parameters

- `nIndex`: the skin index.

int **GetSkin** ()
: return the current skin index. the default skin is at index 0.

void **LoadStoredModel** (int *nModelSetID*)
: Load a stored model in data base by the model set ID. A stored model usually contain the attachments and equipments, but not the base model. This function is only valid when the base model has already been set.

  Parameters

  - `nModelSetID`: the ID of the model set in the database. Some ID may be reserved for user-specified model

void **PlayAnimation** (**const** object *&anims*)
: play a specified animation.

  Parameters

  - `anims`:
    - it can be int of animation ID(external animation id is supported) see local nAnimID = *ParaAsset.CreateBoneAnimProvider*(-1, filename, filename, false);
    - it can also be string of animation file name
    - it can also be a table of {animID, animID}: currently only two are supported. The first one is usually a non-loop, and second one can be loop or non-loop.

int **GetAnimID** ()
: get the current animation ID of the character. Usually 0-46 is for normal animation like idle and walking; 0-1000 are reserved for internally animation. 1000-2000 are game specific; 2000 plus are automatically generated. One should call *GetAnimFileName()* for id above 2000.

  Return : it may return -1 if invalid.

const char ***GetAnimFileName** ()
: get the current animation's filename. If it is an internal animation, it will return nil. If it is from bone animation provider, it returns the file name from which the animation is loaded.

void **EnableAnimIDMap** (bool *bEnable*)
: set whether the m_mapAnimIDs will be used. Disabled by default

bool **IsAnimIDMapEnabled** ()
: get whether the m_mapAnimIDs will be used.Disabled by default

bool **AddAnimIDMap** (int *nFromID*, int *nToID*)
: get whether the m_mapAnimIDs will be used.Disabled by default animation ID mapping is a mapping from one animation ID to another ID. This mapping is usually not used (empty). However, it is used when we can want to secretly replace animation used by this animation instance, by adding a ID to ID map. e.g. we can replace female walk animation ID with an external female walk animation ID.

  Note : remember to *EnableAnimIDMap()* in order for the mapping to take effect.

  Parameters

  - `nFromID`:
  - `nToID`: if it is negative, the mapping nFromID is removed.

void **ClearAllAnimIDMap** ()
: remove all mapping.

bool **HasMountPoint** (int *nMountPointID*)
: whether the character has a mount point at the given ID. Such characters are usually cars, horses, planes, etc.

  Parameters

  - `nMountPointID`: this is usually 0.

bool **IsMounted**()
whether the object is mounted on another object.

void **MountOn_**(**const** char **sTarget*)
mount the current object on another object.
Parameters
- `sTarget`: the name of the global target object on which the current character is mounted on.

void **MountOn**(*ParaObject* &*target*)
the target must contain at least one mount point. if there are multiple mount point, it will mount to the closet one.
Parameters
- `target`: the name of the target object on which the current character is mounted on.

void **MountOn2**(*ParaObject* &*target*, int *nMountID*)
the target must contain at least one mount point. if there are multiple mount point, it will mount to the closet one.
Parameters
- `target`: the name of the target object on which the current character is mounted on.
- `nMountID`: the attachment ID of the model. if -1 (default), we will attach to the nearest mount position.

void **UnMount**()
this will force unmount the characer. However, it is usually not natural to do this explicitly, since we do not know how the character should behave after mount. Instead, one can call player_obj:ToCharacter():AddAction(action_table.ActionSymbols.S_JUMP_START) to unmount by jumping off.

void **SetBodyParams**(int *skinColor*, int *faceType*, int *hairColor*, int *hairStyle*, int *facialHair*)
Set the character body parameters. Need to call *RefreshModel()* after finished with the settings. All body parameters are integer ranging from 0 to maxType, where maxType is the maximum number of types of of a certain body parameter. For each body parameter, one can specify -1 to retain its current value. This is useful, when the caller only wants to change one or several of the body parameters. The default body parameters is (0,0,0,0,0).

int **GetBodyParams**(int *type*)
Get the character body parameters. @ param type: the body parameter type of the character BP_SKINCOLOR =0, BP_FACETYPE = 1, BP_HAIRCOLOR = 2, BP_HAIRSTYLE = 3, BP_FACIALHAIR = 4

void **SetDisplayOptions**(int *bShowUnderwear*, int *bShowEars*, int *bShowHair*)
Set the display options for the character. in case of boolean input: 1 stands for true; 0 stands for false, -1 stands for retaining its current value. Need to call *RefreshModel()* after finished with the settings.

bool **GetDisplayOptions**(int *type*)
Get the display option parameters. @ param type: the display option parameter of the character DO_SHOWUNDERWEAR =0, DO_SHOWEARS = 1, DO_SHOWHAIR = 2

void **SetCharacterSlot**(int *nSlotID*, int *nItemID*)
set the model ID of a specified character slot. Need to call *RefreshModel()* after finished with the settings.
Parameters
- `nSlotID`: the ID of the slot to be set for the character. Normally there are 16 slots on the character. CS_HEAD =0, CS_NECK = 1, CS_SHOULDER = 2, CS_BOOTS = 3, CS_BELT = 4, CS_SHIRT = 5, CS_PANTS = 6, CS_CHEST = 7, CS_BRACERS = 8, CS_GLOVES = 9, CS_HAND_RIGHT = 10, CS_HAND_LEFT

= 11, CS_CAPE = 12, CS_TABARD = 13, CS_FACE_ADDON = 14, // newly added by andy 2009.5.10, Item type: IT_MASK 26 CS_WINGS = 15, // newly added by andy 2009.5.11, Item type: IT_WINGS 27 CS_ARIES_CHAR_SHIRT = 16, // newly added by andy 2009.6.16, Item type: IT_WINGS 28 CS_ARIES_CHAR_PANT = 17, CS_ARIES_CHAR_HAND = 18, CS_ARIES_CHAR_FOOT = 19, CS_ARIES_CHAR_GLASS = 20, CS_ARIES_CHAR_WING = 21, CS_ARIES_PET_HEAD = 22, CS_ARIES_PET_BODY = 23, CS_ARIES_PET_TAIL = 24, CS_ARIES_PET_WING = 25, CS_ARIES_CHAR_SHIRT_TEEN = 28, // newly added by andy 2011.8.13, Item type: IT_ARIES_CHAR_SHIRT_TEEN 40

- `nItemID`: the ID of the item to be put into the character slot. The default value for all slots is 0. One may empty a certain slots by setting its nItemID to 0.

int **`GetCharacterSlotItemID`**(int *nSlotID*)
Get the model ID of a specified character slot.
Return : the item ID on the given slot. 0 stands for empty. -1 if invalid slot ID
Parameters

- `nSlotID`: the ID of the slot to be set for the character. Normally there are 16 slots on the character.

void **`LoadFromFile`**(**const** char **filename*)
load all settings of the model to file. Need to call *RefreshModel()* after loading.

void **`SaveToFile`**(**const** char **filename*)
save all settings of the model to file.

void **`RefreshModel`**()
update the entire model from its characters settings, such as body parameters and equipments. This will rebuild the composed character texture.

void **`UseAIModule`**(**const** char **sAIType*)
use a specified AI object.
Parameters

- `sAIType`: valid AI object is: “NPC”|”“|”NULL” “” is the same as “NPC” “NULL” means no AI module.

void **`AssignAIController`**(**const** char **sAICtrlerName*, **const** char **sParam1*)
assign a new controller to the current AI object. if there is no AI object, we will create a default one to use. sAICtrlerName = “face”: Face tracking controller: sParam1 = “true”|”false”: “true” to enable face tracking. e.g. Char:AssignAIController(“face”, true);
Parameters

- `sAICtrlerName:valid`: AI controller name is: “sequence”|”movie”|”face”|”follow”|”avoid”
- `sParam1`: this format of this parameter is dependent on the sAICtrlerName. see below:

sAICtrlerName = “follow”: follow another named biped: sParam1 = “” | “sName” | “sName radius angle” : “” to disable following, or follow a biped called sName.

- sName: the name of the biped to follow,
- radius: [optional, default to 2.5f] it is the default radius around the target biped. it will control the biped to try it best to stand on this circle.
- angle: [optional, default to Pi] it will control the biped to stand beside the target with the target facing shifted by this value. note that +-Pi means behind the biped;0 means in front of the character. e.g. “playername”, “playername 2.5 3.14”, “playername 3.0 0”, “playername 3.0 1.57”, “playername 3.0 -1.57” e.g. Char:AssignAIController(“follow”, “player1”); e.g. Char:AssignAIController(“follow”, “player1 2.5 3.14”);

sAICtrlerName = “movie”: enable a movie controller. sParam1 = “”|”true”|”false”: “” or “true” to enable a movie, or “false” to disable it. e.g. Char:AssignAIController(“movie”, “true”);Char:AssignAIController(“movie”, “false”); use *GetMovieController()* to get the controller sAICtrlerName = "sequence": enable a sequence controller. sParam1 = ""|"true"|"false": "" or "true" to enable a sequence, or "false" to disable it. e.g. Char:AssignAIController("sequence", "true");Char:AssignAIController("sequence", "false"); use *GetSeqController()* to get the controller

See : *UseAIModule()*

bool **`IsAIControllerEnabled`** (**const** char **sAICtrlerName*)

whether a certain controller is enabled.

Parameters

- `sAICtrlerName`: "sequence"|"movie"|"face"|"follow"|"avoid" see also *AssignAIController()*;

ParaMovieCtrler **`GetMovieController`** ()

get the movie controller. the movie controller will be created if it does not exist.

ParaSeqCtrler **`GetSeqController`** ()

get the sequence controller. the sequence controller will be created if it does not exist.

ParaFaceTrackingCtrler **`GetFaceTrackingController`** ()

get the face tracking controller. the sequence controller will be created if it does not exist.

void **`CastEffect`** (int *nEffectID*)

cast a magic effect by the effect ID.

Parameters

- `nEffectID`: effect ID in the effect database if effect id is negative, the effect will be removed from the object.

void **`CastEffect2`** (int *nEffectID*, **const** char **sTarget*)

cast a magic effect by the effect ID.

Parameters

- `nEffectID`: effect ID in the effect database if effect id is negative, the effect will be removed from the object.
- `sTarget`: target of the effect, if there is any. it can either to the name of the target object, or an attachment ID on the current character. in case it is an attachment ID. It should be in the following format. <d> d is the attachment ID. For customizable characters, some IDs are: 0 ATT_ID_SHIELD, 1 ATT_ID_HAND_RIGHT, 2 ATT_ID_HAND_LEFT, default value 5 ATT_ID_SHOULDER_RIGHT, 6 ATT_ID_SHOULDER_LEFT, 11 ATT_ID_HEAD, e.g. char:CastEffect(1, "NPC0"); fire missile 1 at NPC0. char:CastEffect(2, "<2>"); attach the effect 2 to the current character's hand.

void **`AddAttachment`** (*ParaAssetObject* *ModelAsset*, int *nAttachmentID*)

Parameters

- `ModelAsset`: the model to be attached. This can be both ParaX model or static mesh model.
- `nAttachmentID`: to which part of the character, the effect model is attached. ATT_ID_SHIELD = 0, ATT_ID_HAND_RIGHT = 1, ATT_ID_HAND_LEFT = 2, ATT_ID_TEXT = 3, ATT_ID_GROUND = 4, ATT_ID_SHOULDER_RIGHT = 5, ATT_ID_SHOULDER_LEFT = 6, ATT_ID_HEAD = 11, ATT_ID_FACE_ADDON = 12, ATT_ID_EAR_LEFT_ADDON = 13, ATT_ID_EAR_RIGHT_ADDON = 14, ATT_ID_BACK_ADDON = 15, ATT_ID_WAIST = 16, ATT_ID_NECK = 17, ATT_ID_BOOTS = 18, ATT_ID_MOUTH = 19, ATT_ID_MOUNT1 = 20,
- `nSlotID`: the slot id of the effect. default value is -1. if there is already an effect with the same ID it will be replaced with this new one.
- `scaling`: scaling of the texture
- `ReplaceableTexture`: replace the texture.

ParaScripting::*ParaAttributeObject* **`GetAttachmentAttObj`** (int *nAttachmentID*)

get the attachment object's attribute field.

void **RemoveAttachment** (int *nAttachmentID*)
: the model to be detached.

See *AddAttachment()*;

Parameters

- `nAttachmentID`: this value is reserved and can be any value.
- `nSlotID`: the slot id of the effect. default value is -1. all attachments with the SlotID will be removed.

void **Stop** ()
: stop the biped if it is moving.

void **MoveTo** (double *x*, double *y*, double *z*)
: move (using the current style i.e. walk or run) to a position relative to the current position.

void **MoveAndTurn** (double *x*, double *y*, double *z*, float *facing*)
: move (using the current style i.e. walk or run) to a position relative to the current position and turn.

double **GetCartoonFaceComponent** (int *nComponentID*, int *SubType*)
: get the cartoon face associated with this object. CFS_TOTAL_NUM, }; SubType:

•0: style: int [0,00]
•1: color: 32bits ARGB
•2: scale: float in [-1,1]
•3: rotation: float in (-3.14,3.14]
•4: x: (-128,128]
•5: y: (-128,128]
Note : check *IsSupportCartoonFace()* before using this function

Parameters

- `nComponentID`: one of the following value. cartoon face slots on the face of the character enum CartoonFaceSlots { CFS_FACE = 0, CFS_WRINKLE = 1, CFS_EYE = 2, CFS_EYEBROW = 3, CFS_MOUTH = 4, CFS_NOSE = 5, CFS_MARKS = 6,

string **GetCartoonFaceComponentCode** (int *nComponentID*, int *SubType*)
: same as GetCartoonFaceComponent, except that the returned value is a string code.

Parameters

- `nComponentID`: enum CartoonFaceSlots, in most cases, this is CFS_FACE(0). SubType:
  - 0: style: int [0,00]: code is the string presentation of int.
  - 1: color: 32bits ARGB: code is hex presentation of RGB of the value, such as “ffffff”
  - 2: scale: float in [-1,1]: code is one hex number, where “0” is 0, “9” is 9/9, “a” is -1/7, “f” is -7/7.
  - 3: rotation: float in (-3.14,3.14]: code is at most 2 hex number, where “0” is 0.
  - 4: x: (-128,128]: code is at most 2 hex number, where “0” is 0.
  - 5: y: (-128,128]: code is at most 2 hex number, where “0” is 0. return the code usually hex format, such as “ffffff”

void **SetCartoonFaceComponent** (int *nComponentID*, int *SubType*, double *value*)
: see *GetCartoonFaceComponent()*

Note : check *IsSupportCartoonFace()* before using this function

void **SetCartoonFaceComponentCode** (int *nComponentID*, int *SubType*, **const** char **color*)
: see *GetCartoonFaceComponentCode()*

Note : check *IsSupportCartoonFace()* before using this function

bool **IsSupportCartoonFace** ()
: check whether the associated cartoon model supports cartoon face.

void **SetSkinColorMask**(**const** char **strColor*)
the color mask to be applied to cartoon face and ccs base layer. It can be used to fine tune skin color on top of exiting base skin and face textures. Default value is "ffffff". setting this to a different value will degrade the performance a little.
Parameters
- `strColor`: the hex string of RGB, such as "RGB".

string **GetSkinColorMask**()
the color mask to be applied to cartoon face and ccs base layer. It can be used to fine tune skin color on top of exiting base skin and face textures. Default value is "ffffff". setting this to a different value will degrade the performance a little.
Return the hex string of RGB, such as "fffff"

int **GetGender**()
get the character gender.
Return : 0 if male, 1 if female, -1 if error

int **GetRaceID**()
get the character race ID according to CharRacesDB table
Return : race ID, -1 if error

class ParaDataProvider
#include <ParaScriptingWorld.h> Wrapper of internal *ParaWorld* data provider

Public Functions

bool **IsValid**()
check if the object is valid

bool **InsertPuzzleRecordFromString**(**const** char **strRecord*)
Insert the new puzzle record to Puzzle_DB
Return true if the record is inserted in database
Parameters
- `record`: ID of record will be ignored and filled with actual ID if inserted successfully

bool **DeletePuzzleRecordByID**(int *ID*)
delete the existing puzzle record from Puzzle_DB
Return true if the record is deleted in database
Parameters
- `ID`: ID of target record

int **GetNPCIDByName**(**const** char **name*)
get NPC ID by name
Return id is returned if found in database; otherwise non-positive value(-1) is returned.
Parameters
- `name`: name of character

int **GetNPCCount**()
the total number of NPC in the database. There is no statement cached for this call.
Return

bool **DoesAttributeExists**(**const** char **sName*)
whether a given attribute exists. This function is usually used internally. One common use of this function is to test if a field exists, so that we know if a object has been saved before or not.
Return

Parameters

- `sName`: : name of the attribute field

object **`GetAttribute`**(**const** char **sName*, **const** object &*sOut*)

get the attribute of the current attribute instance in the current table e.g. suppose db is a *ParaDataProvider* object. local x = db:GetAttribute("PlayerX", 0); local name = db:GetAttribute("PlayerName", "no_name");

Return the object is returned. Currently it may be a string, boolean or a number.

Parameters

- `sName`: : name of the attribute field
- `sOut`: : [in|out] default value of the field. Currently it may be a string or a number

bool **`UpdateAttribute`**(**const** char **sName*, **const** object &*sIn*)

update the attribute of the current attribute instance in the current table insert the attribute if it is not created before.

Return true if succeed.

Parameters

- `sName`: : name of the attribute field
- `sIn`: value of the attribute field. Currently it may be a string, boolean or a number.

bool **`InsertAttribute`**(**const** char **sName*, **const** object &*sIn*)

insert by simple name value pair

Return true if succeed.

Parameters

- `sName`: : name of the attribute field
- `sIn`: value of the attribute field. Currently it may be a string, boolean or a number.

bool **`DeleteAttribute`**(**const** char **sName*)

delete the attribute of the current attribute instance in the current table

Return true if succeed.

Parameters

- `sName`: : name of the attribute field

bool **`ExecSQL`**(**const** char **sCmd*)

run a given sql command, commonly used commands are "BEGIN", "END", when we are batch saving attributes.

void **`SetTableName`**(**const** char **sName*)

set current attribute table name, the table will be created if not exists.

Parameters

- `sName`: table name

const char ***`GetTableName`**()

get the current attribute table name

Return current attribute table name

class **`ParaFaceTrackingCtrler`**

#include <ParaScriptingCharacter.h> *ParaFaceTrackingCtrler* object: it will allow the biped to always face to a given target or another biped.

Public Functions

bool **`IsValid`**()

check if the object is valid

void **FaceTarget** (float *x*, float *y*, float *z*, float *fDuration*)
instruct a character to face a target for a certain duration. It will smoothly rotate the character neck to face it

Parameters

- `xyz`: which point in world space to face to
- `fDuration`: how many seconds the character should keeps facing the target. Set zero or a negative value to cancel facing previous point.

class ParaFileObject
#include <ParaScriptingIO.h> file object. Always call *close()* when finished with the file.

Public Functions

bool **IsValid** ()
whether file is valid

void **close** ()
Close the current file.

int **SetSegment** (int *nFromByte*, int *nByteCount*)
by setting the segment, we can inform NPL that we only transmit the file content in the segment to a remote place. it also affects the returned result of the GetBase64String.

Return : return the number of bytes in the segment. it is usually the same as nByteCount, unless nBytesCount exceeds the file length.

Parameters

- `nFromByte`: from which byte
- `nByteCount`: number of bytes of the segment. if -1, it is end of file.

const char ***GetBase64StringEx** (int **pnStrLength* = 0)
get base 64 string of the binary data in the current file segment. to change segment, use SetSegment.

Return pOutString: a static global buffer containing the string. it ends with '\0'. it is NULL, if file is invalid.

Parameters

- `pnStrLength`: [out] length of the returned string.

void **seek** (int *offset*)
always call seek(0) if one opens a old file for overwritten

int **getpos** ()
get current reader or writer cursor position offset in bytes

void **SetFilePointer** (int *lDistanceToMove*, int *dwMoveMethod*)
The SetFilePointer function moves the file pointer of an open file. this function only works when the ParaFile object is an actual windows file, instead of a virtual file. for virtual file, use the seek and seekRelative function.

Parameters

- `lDistanceToMove`:
- `dwMoveMethod`: 0: FILE_BEGIN The starting point is 0 (zero) or the beginning of the file. 1: FILE_CURRENT The starting point is the current value of the file pointer. 2: FILE_END The starting point is the current end-of-file position.

bool **SetEndOfFile** ()
The SetEndOfFile function moves the end-of-file (EOF) position for the specified file to the current position of the file pointer.This function can be used to truncate or extend a file. If the file is extended, the contents of the file between the old EOF position and the new position are not defined.

Return

const char ***`readline`**()
read line as a string. The string is guaranteed to be ended with '\0'. if end of file is reached, it will return NULL. which is nil in the script. if a line begins with "–", it is automatically recognized as a comment line and will be skipped. a blank line will also be skipped.

const char ***`GetText`**()
get the content of the file as text. Text encoding is escaped. If you want to get the raw file text with the heading BOM, such as utf8 (EE BB BF), use GetText2(0,-1)

const std::string &**`GetText2`**(int *fromPos*, int *nCount*)
get the content of the file as text between two positions.
Parameters
- `fromPos`: position in bytes.
- `nCount`: nCount in bytes. if -1, it defaults to end of file.

const std::string &**`ReadString`**(int *nCount*)
read a binary string of length nCount from current position.
Parameters
- `nCount`: nCount in bytes. if -1, it defaults to end of file.

void **`WriteString`**(**const** char **str*)
write a string to the current file.

void **`WriteString2`**(**const** char **buffer*, int *nSize*)
write a buffer to the current file.

void **`write`**(**const** char **buffer*, int *nSize*)
write a buffer to the current file.

int **`WriteBytes`**(int *nSize*, **const** object &*input*)
write bytes to file; e.g. local nBytes = file:WriteBytes(3, {[1]=255, [2]=0, [3]=128});
Return : number of bytes written
Parameters
- `nSize`: number of bytes to write
- `input`: array of integers, each represents a byte

object **`ReadBytes`**(int *nSize*, **const** object &*output*)
read bytes from file. e.g. local data={};local nBytes = file:ReadBytes(3, data); data[1], data[2], data[3]
Return string or a table array
Parameters
- `nSize`: number of bytes to read. if negative, it will be total file size
- `output`: in/out, it should be an empty table. When the function returns, it will contain an array of integers, each represents a byte if output is nil or "", the returned value will also be a string(may contain \0) of the content read

void **`WriteFloat`**(float *value*)
float is 32 bits

void **`WriteWord`**(int *value*)
integer is converted 16 bits unsigned word

void **`WriteInt`**(int *value*)
integer is 32 bits

void **`WriteShort`** (int *value*)
integer is 16 bits signed int

void **`WriteUInt`** (unsigned int *value*)
integer is 32 bits unsigned

int **`GetFileSize`** ()
get the file size in bytes.

class **`ParaFileSystemWatcher`**
#include <ParaScriptingIO.h> file system watcher to monitor file changes in one or several directories .NET also provides a class similar to this one, called FileSystemWatcher.

Public Functions

void **`AddDirectory`** (**const** char **filename*)
add a directory to monitor

void **`RemoveDirectory`** (**const** char **filename*)
remove a directory from monitor

void **`AddCallback`** (**const** char **sCallbackScript*)
the call back script will be invoked with a global msg msg = {type=[0,5], dirname=string, filename=string}, where type can be { null = 0, added = 1, removed = 2, modified = 3, renamed_old_name = 4, renamed_new_name = 5 };

class **`ParaGlobal`**
#include <ParaScriptingGlobal.h> *ParaGlobal* namespace contains a list of HAPI functions to globally control the engine

Public Static Functions

void **`ExitApp`** ()
exit the applications.

void **`Exit`** (int *nReturnCode*)
This is same as ExitApp, except that it supports a return code. this is the recommended way of exiting application. this is mainly used for writing test cases. Where a return value of 0 means success, any other value means failure.

void **`WriteToConsole`** (**const** char **strMessage*)
write const char* to console, usually for debugging purposes.

void **`WriteToLogFile`** (**const** char **strMessage*)
write const char* to log file, usually for debugging purposes.

int **`GetLogPos`** ()
get the current log file position. it is equivalent to the log file size in bytes. one can later get log text between two Log positions.

const char ***`GetLog`** (int *fromPos*, int *nCount*)
get log text between two Log positions.
Return string returned.
Parameters

- `fromPos`: position in bytes. if nil, it defaults to 0
- `nCount`: count in bytes. if nil, it defaults to end of log file.

ParaScripting::*ParaServiceLogger* **GetLogger** (**const** object &*name*)
Get a service logger. Please see util/LogService.h for more information.

bool **WriteToFile** (**const** char **filename*, **const** char **strMessage*)
write const char* to specific file. obsolete

void **SetGameStatus** (**const** char **strState*)
set the game status
Parameters
- `strState`:
  - "disable" disable the game
  - "enable" enable the game
  - "pause" pause the game
  - "resume" resume the game

double **GetGameTime** ()
Return return the current game time in milliseconds.When game is paused, game time is also paused. this is usually used for cinematic movies

std::string **GetDateFormat** (**const** object &*sFormat*)
get the date in string [thread safe]
Parameters
- `sFormat`: can be NULL to use default.e.g. "ddd',' MMM dd yy"
  - d Day of month as digits with no leading zero for single-digit days.
  - dd Day of month as digits with leading zero for single-digit days.
  - ddd Day of week as a three-letter abbreviation. The function uses the LOCALE_SABBREVDAYNAME value associated with the specified locale.
  - dddd Day of week as its full name. The function uses the LOCALE_SDAYNAME value associated with the specified locale.
  - M Month as digits with no leading zero for single-digit months.
  - MM Month as digits with leading zero for single-digit months.
  - MMM Month as a three-letter abbreviation. The function uses the LOCALE_SABBREVMONTHNAME value associated with the specified locale.
  - MMMM Month as its full name. The function uses the LOCALE_SMONTHNAME value associated with the specified locale.
  - y Year as last two digits, but with no leading zero for years less than 10.
  - yy Year as last two digits, but with leading zero for years less than 10.
  - yyyy Year represented by full four digits.
  - gg Period/era string. The function uses the CAL_SERASTRING value associated with the specified locale. This element is ignored if the date to be formatted does not have an associated era or period string.

std::string **GetTimeFormat** (**const** object &*sFormat*)
get the time in string [thread safe]
Parameters
- `sFormat`: can be NULL to use default. e.g. "hh':'mm':'ss tt"
  - h Hours with no leading zero for single-digit hours; 12-hour clock.
  - hh Hours with leading zero for single-digit hours; 12-hour clock.
  - H Hours with no leading zero for single-digit hours; 24-hour clock.
  - HH Hours with leading zero for single-digit hours; 24-hour clock.
  - m Minutes with no leading zero for single-digit minutes.
  - mm Minutes with leading zero for single-digit minutes.
  - s Seconds with no leading zero for single-digit seconds.

- ss Seconds with leading zero for single-digit seconds.
- t One character time-marker string, such as A or P.
- tt Multicharacter time-marker string, such as AM or PM.

DWORD **timeGetTime**()

The timeGetTime function retrieves the system time, in milliseconds. The system time is the time elapsed since Windows was started. Note that the value returned by the timeGetTime function is a DWORD value. The return value wraps around to 0 every 2^32 milliseconds, which is about 49.71 days. This can cause problems in code that directly uses the timeGetTime return value in computations, particularly where the value is used to control code execution. You should always use the difference between two timeGetTime return values in computations.

double **getAccurateTime**()

get the elapsed time using high-resolution timing function in seconds. this function is mostly used for profiling on the NPL

double **random**()

the random seed is set at application start

Return generate a random number between [0,1]

double **GetSysDateTime**()

Get the system date and time in seconds. The system time is expressed in Coordinated Universal Time (UTC). Note: there is some trick to make the returned value a valid number in NPL. Only compare time with time returned by the same function. TODO: in the long run, a true unsigned int64 should be returned. [thread safe]

std::string **GenerateUniqueID**()

generate a unique ID as a string. This is usually a string. [thread safe]

Return

void **SaveObject**(**const** char **strObjectName*, **const** object *&objObject*)

global object dictionary functions: this is a way for different script runtime to share some global information. Currently only value and const char* object can be saved. one can save nil to a object name to delete the object.

Parameters

- `objObject`: object to save

object **LoadObject**(**const** object *&strObjectName*)

global object dictionary functions: this is a way for different script runtime to share some global information. Currently only value and const char* object can be saved. return nil, if object is not found

Parameters

- `strObjectName`: the object name

void **SetGameLoop**(**const** char **scriptName*)

reset the game loop script. the game loop script will be activated every 0.5 seconds see *SetGameLoopInterval()* to change the default interval Please keep the game loop concise. The default game loop is ./script/gameinterface.lua

void **SetGameLoopInterval**(float *fInterval*)

set the game loop activation interval. The default value is 0.5 seconds.

bool **CreateProcess**(**const** char **lpApplicationName*, **const** char **lpCommandLine*, bool *bWaitOnReturn*)

run an external application. creates a new process and its primary thread. The new process runs the specified executable file in the security context of the calling process.

Remark : One can also use ParaEngine C++ or .Net API to write application plug-ins for the game engine, which can be loaded like any other script files. e.g. To open a file in an external notepad editor use *ParaGlobal.CreateProcess*("c:\\notepad.exe", "\"c:\notepad.exe" c:\test.txt", true);

Return true if opened.

Parameters

- `lpApplicationName:Pointer`: to a null-terminated string that specifies the module to execute. The specified module can be a Windows-based application. The string can specify the full path and file name of the module to execute or it can specify a partial name. In the case of a partial name, the function uses the current drive and current directory to complete the specification. The function will not use the search path. If the file name does not contain an extension, .exe is assumed. If the executable module is a 16-bit application, lpApplicationName should be NULL, and the string pointed to by lpCommandLine should specify the executable module as well as its arguments.
- `lpCommandLine:Pointer`: to a null-terminated string that specifies the command line to execute.
- `bWaitOnReturn`: if false, the function returns immediately; otherwise it will wait for the editor to return. if this is true, the Child Process will have Redirected Input and Output to current log file.

bool **`ShellExecute`**(**const** char **lpOperation*, **const** char **lpFile*, **const** char **lpParameters*, **const** char **lpDirectory*, int *nShowCmd*)

Performs an operation on a specified file. e.g. *ParaGlobal.ShellExecute*("open", "iexplore.exe", "http://www.paraengine.com", nil, 1);

Parameters

- `lpOperation:[in]`: Pointer to a null-terminated string,
  - "wait" this is a special one that uses ShellExecuteEx to wait on the process to terminate before return
  - "edit" Launches an editor and opens the document for editing. If lpFile is not a document file, the function will fail.
  - "explore" Explores the folder specified by lpFile.
  - "find" Initiates a search starting from the specified directory.
  - "open" Opens the file specified by the lpFile parameter. The file can be an executable file, a document file, or a folder.
  - "print" Prints the document file specified by lpFile. If lpFile is not a document file, the function will fail.
  - NULL For systems prior to Microsoft Windows 2000, the default verb is used if it is valid and available in the registry. If not, the "open" verb is used.
- `lpFile`: [in] Pointer to a null-terminated string that specifies the file or object on which to execute the specified verb. To specify a Shell namespace object, pass the fully qualified parse name. Note that not all verbs are supported on all objects. For example, not all document types support the "print" verb.
- `lpParameters[in]`: If the lpFile parameter specifies an executable file, lpParameters is a pointer to a null-terminated string that specifies the parameters to be passed to the application. The format of this string is determined by the verb that is to be invoked. If lpFile specifies a document file, lpParameters should be NULL.
- `lpDirectory`: [in] Pointer to a null-terminated string that specifies the default directory.
- `nShowCmd`: [in] Flags that specify how an application is to be displayed when it is opened. If lpFile specifies a document file, the flag is simply passed to the associated application. It is up to the application to decide how to handle it.
  - #define SW_HIDE 0
  - #define SW_NORMAL 1
  - #define SW_MAXIMIZE 3
  - #define SW_SHOW 5
  - #define SW_MINIMIZE 6

    – #define SW_RESTORE 9

bool **OpenFileDialog**(**const** object *&inout*)
create a open file dialog. This function does not return until the user selects a dialog.
Return : true if user clicks ok. and the inout.filename contains the result.
Parameters
- `inout`: input table:{filter="All Files (*.*);*.*;", filterindex, initialdir, flags, } t.filter="All Files (*.*)\0*.*\0" output : {filename, result=true} t.filename: the full path and file name specified by the user t.result: boolean if user clicks the OK button

bool **WriteRegStr**(**const** string *&root_key*, **const** string *&sSubKey*, **const** string *&name*, **const** string *&value*)
Write a string to the registry. e.g. WriteRegStr("HKLM", "Software\My Company\My Software", "string Value", "string Name");
Parameters
- `root_key`: must be HKCR or HKEY_CLASSES_ROOT HKLM or HKEY_LOCAL_MACHINE HKCU or HKEY_CURRENT_USER HKU or HKEY_USERS

const char ***ReadRegStr**(**const** string *&root_key*, **const** string *&sSubKey*, **const** string *&name*)
Read string from the registry. Valid values for root_key are listed under WriteRegStr. NULL will be returned if the string is not present. If the value is present, but is of type REG_DWORD, it will be read and converted to a string.
Parameters
- `root_key`: must be HKCR or HKEY_CLASSES_ROOT HKLM or HKEY_LOCAL_MACHINE HKCU or HKEY_CURRENT_USER HKU or HKEY_USERS

bool **WriteRegDWORD**(**const** string *&root_key*, **const** string *&sSubKey*, **const** string *&name*, DWORD *value*)
Write a DWORD to the registry. see *WriteRegStr()* for more info
Parameters
- `root_key`: must be HKCR or HKEY_CLASSES_ROOT HKLM or HKEY_LOCAL_MACHINE HKCU or HKEY_CURRENT_USER HKU or HKEY_USERS

DWORD **ReadRegDWORD**(**const** string *&root_key*, **const** string *&sSubKey*, **const** string *&name*)
Read DWORD from the registry. Valid values for root_key are listed under WriteRegStr. NULL will be returned if the DWORD is not present or type is a string.
Parameters
- `root_key`: must be HKCR or HKEY_CLASSES_ROOT HKLM or HKEY_LOCAL_MACHINE HKCU or HKEY_CURRENT_USER HKU or HKEY_USERS

class ParaHTMLBrowser
#include <ParaScriptingHTMLBrowser.h> a HTML browser control and texture

Public Functions

bool **IsValid**()
check if the object is valid

const char ***GetName**()
get the window name.

const char ***GetLastNavURL**()
the last url when navigateTo() is called. default is ""

void **Release**()
something like delete this

class **ParaIO**

#include <ParaScriptingIO.h> *ParaIO* class: IO functions ported to the scripting system

Public Static Functions

const char ***DecodePath**(**const** char **input*)

replace variables in input path and return the result path. see AddPathVariable

Return the resulting path. Please note that the return value is the input itself if nothing is replaced. otherwise, it is a static string reference containing the result. therefore the result is NOT thread safe.

Parameters

- `input`: a path with or without replaceable. make sure you have called *ToCanonicalFilePath()* to canonicalize the input before calling this function

const char ***EncodePath**(**const** char **input*)

this does reverse of DecodePath. see AddPathVariable

Parameters

- `input`: a path with or without replaceable. make sure you have called *ToCanonicalFilePath()* to canonicalize the input before calling this function

const char ***EncodePath2**(**const** char **input*, **const** char **varNames*)

same as EncodePath, except that it will only replace variables who name appears in varNames. varNames is a comma separated list of variable names.

Parameters

- `varNames`: a comma separated list of variable names. like "WORLD,USERID", etc.

bool **AddPathVariable**(**const** char **sVarName*, **const** char **sVarValue*)

add a new variable to the replaceable pool

Return : true if succeed. it may return false, if a protected variable with the same name already exist but it not editable via scripting interface.

Parameters

- `sVarName`: the variable name without enclosing %%, such as "WORLD", "USERID", usually uppercase.
- `sVarValue`: the path that the variable expands to. If nil, it will remove the variable.

bool **AddSearchPath**(**const** char **sFile*)

add a search path to the search path pool. It will internally normalize the path and check for duplicates note: this function shall not be called by an untrusted client, since it will secretly swap files. : shall we support remote http zip file as a valid search path?

bool **RemoveSearchPath**(**const** char **sFile*)

remove a search path from the search path pool.

std::string **GetWritablePath**()

get writable path

bool **ClearAllSearchPath**()

clear all search paths.

unsigned long **CRC32**(**const** char **filename*)

Get the CRC 32 code of a given file.

Return : return 0 if not succeed, otherwise the CRC32 code is returned.

void **UpdateMirrorFiles**(**const** char **dirName*, bool *bOverwrite*)

this function is equivalent to calling the following functions. LoadLogFromFile("InstallFiles.txt"); LoadLogFromFile("temp/filelog.txt"); MirrorFiles(dirName, bOverwrite);

e.g. UpdateMirrorFiles("_InstallFiles/", true);

Parameters

- `dirName`: such as "_InstallFiles/"
- `bOverwrite`: if this is true, existing files will be overridden.

ParaScripting::*ParaZipWriter* **`CreateZip`**(**const** char **fn*, **const** char **password*)

call this to start the creation of a zip file.

int **`DeleteFile`**(**const** char **sFilePattern*)

delete a given file. It will reject any system files outside the application directory. after all, this function is of high security level.

Return : the number of files deleted.

Parameters

- `sFilePattern`: such as "*.dds", "temp.txt", etc

bool **`MoveFile`**(**const** char **src*, **const** char **dest*)

The MoveFile function will move (rename) either a file or a directory (including its children) either in the same directory or across directories.

Return true if succeeds

Parameters

- `src`: specifies the name of an existing file
- `dest`: specifies the name of the new file

bool **`CopyFile`**(**const** char **src*, **const** char **dest*, bool *bOverride*)

The CopyFile function copies an existing file to a new file

Return true if succeeds

Parameters

- `src`: specifies the name of an existing file
- `dest`: specifies the name of the new file
- `bOverride`: [in] If this parameter is false and the new file specified by src already exists, the function fails. If this parameter is true and the new file already exists, the function overwrites the existing file and succeeds.

bool **`CreateNewFile`**(**const** char **filename*)

create a new file for writing. it will make all necessary directories in order to create the file.

bool **`OpenFileWrite`**(**const** char **filename*)

open a new file for write-only access. If the file does not exist, it will be created. if the file exists, the file pointer is at the end of file.

bool **`OpenFile`**(**const** char **filename*)

Open a file for read-only access.

ParaFileObject **`OpenAssetFile`**(**const** char **filename*)

This is rather similar to *OpenFile()* method, except that it will first look in the AssetManifest to see if the file exit. If the file does appear in manifest list, we will download the latest version from the current asset server, if not done before. the download process is SYNCHRONOUS. If the file does not appear in AssetManifest list, this function is equivalent to *OpenFile()*

Return : 1 if succeed. 0, if file is not downloaded successfully.

Parameters

- `filename`: the asset file key to open. The actual file opened may from the temp/cache/ folder.
- `bDownloadIfNotUpToDate`: default to true. if true, we will download the latest version from the current asset server. the download process is synchronous. If false, the function will return 0 immediately, when the caller may consider download the file asynchronously, and then open the file again.

bool **DoesAssetFileExist** (**const** char **filename*)

check to see whether we have a up to date version of an asset file. if the asset file does not appear in asset manifest list, it will return the result of *DoesFileExist()* instead.

bool **DoesAssetFileExist2** (**const** char **filename*, bool *bSearchZipFile*)

same as DoesAssetFileExist, except that if bSearchZipFile == false, it is equivalent to *DoesFileExist()*.

int **SyncAssetFile_Async** (**const** char **filename*, **const** char **sCallBackScript*)

similar to SyncFile(), except that this function will return immediately and does not redownload or call AddDownloadCount. And use callback.

Return : 0 if download has begun, 1 if file is already downloaded, -1 if failed, -2 if input is not an asset file.

Parameters

- `sCallBackScript`: the callback script code to be executed when the download is complete. it must begin with ";", such as ";log([[download is complete]]);" the global "msg.res" table contains the error code in case an error is met. msg.res == 0 if succeed, otherwise -1.

int **CheckAssetFile** (**const** char **filename*)

check to see whether an asset file is already downloaded to local disk. generally return value is larger than 1 if succeed.

Return : 1 if already downloaded; 0 if asset has not been downloaded; -1 if we are unable to download the asset file after trying serveral times; -3 if asset is being downloaded but is not completed; -4 if input file is not an asset file.

ParaFileObject **open** (**const** char **filename*, **const** char **mode*)

Open or create a file e.g. *ParaIO.open*("temp/test.txt", "w");

Return file object is returned.

Parameters

- `filename`: the file name to open. if it is "<memory>" and mode is "w". it is a memory buffer.
- `mode`: : access mode
  - "r" Opens for reading. If the file does not exist or cannot be found, the call fails.
  - "w" Opens an empty file for writing. If the given file exists, its contents are destroyed.If not, file will be created.
  - "a" append to the end of an existing file. if file does not exist, a new one is created.

ParaScripting::*ParaFileObject* **openimage** (**const** char **filename*, **const** char **mode*)

open an image file. The r,g,b can then be retrieved as bytes arrays using ReadBytes() function.

Parameters

- `filename`: such as BMP, DDS, JPG, etc. It must be a square image. The size of the image can thus be calculated by file size.
- `mode`: : access mode
  - "r8g8b8": each pixel is a three bytes of R,G,B
  - "a8r8g8b8": each pixel is a four bytes of A,R,G,B
  - "float32": each pixel is a four bytes of float. [Not supported yet]

bool **CreateDirectory** (**const** char **filename*)

make directory

Return : true if the directory is made or already exists

Parameters

- `filename`: director path. file portion will be automatically striped off. So it is ok to pass in file name, instead of directory name.

void **CloseFile** ()

Close the current file.

void **WriteString**(**const** char **str*)
write a string to the current file.

const char ***readline**()
read line as a string. The string is guaranteed to be ended with '\0'. if end of file is reached, it will return NULL. which is nil in the script. if a line begins with "–", it is automatically recognized as a comment line and will be skipped. a blank line will also be skipped.

void **write**(**const** char **buffer*, int *nSize*)
write a buffer to the current file.

bool **DoesFileExist**(**const** char **filename*, bool *bSearchZipFiles*)
Check whether a given file exists on disk.
Parameters
- `filename`: file name to check
- `bSearchZipFiles`: if false, not disk file is searched. If true, both the disk file and zip file will be searched. currently bSearchZipFiles can only be false. Because it is not efficient to check existence of ZIPPED files. Use *OpenFile()* and check for return value instead.

bool **DoesFileExist_**(**const** char **filename*)
see *DoesFileExist()*. This version is same as DoesFileExist(filename, false);

bool **BackupFile**(**const** char **filename*)
backup a specified file, if the file exists. A new file with an extension ".bak" appended to the end of the original file will be created, whose content is identical to the original file.
Return : return true if the file is backed up. return false, if the file does not exist or some error occurs during backup.
Parameters
- `filename`: file name to back up

ParaSearchResult **SearchFiles**(**const** char **sRootPath*, **const** char **sFilePattern*, **const** char **sZipArchive*, int *nSubLevel*, int *nMaxFilesNum*, int *nFrom*)
search files at once.
See CSearchResult the current version of this function can support only one query at a time. The search result is invalid if called intermitantly
Return : one should manually release the search result.
Parameters
- `sRootPath`: the root path. for example: "", "xmodel/","xmodel/models/". Other format is not acceptable
- `sFilePattern`: file pattern, e.g. "*.x" (all files with x extension), "*" (any files), "*."(directories only) if sZipArchive is non-empty, sFilePattern support both regular expression and wild card search. it performs wild cards search by default, where "/\\" matches to directory. "* "matches to anything except "/\\.", and "." matches to "." itself. e.g.. "*.*", "*.", "worlds/ *.abc", "*abc/ *.jpg", etc it sFilePattern begins with ":", things after ":" is treated like a regular expression. It has the same syntax with the perl regular expression and uses full match. e.g.. ":.*\\.jpg", etc.
- `sZipArchive`: it can be "" or a zip archive file name. . if it is not, only that archive files are saved.
  - "": only disk files are searched
  - "*.zip": currently opened zip files are searched
  - "*.*": search disk file followed by all zip files.
- `nSubLevel`: how many sub folders of sRootPath to look into. default value is 0, which only searches the sRootPath folder.
- `nMaxFilesNum`: one can limit the total number of files in the search result. Default value is 50. the search will stop at this value even there are more matching files.
- `nFrom`: only contains results from nFrom to (nFrom+nMaxFilesNum)

string **`GetCurDirectory`**(DWORD *dwDirectoryType*)
get the current directory of the application. it allows querying a number of standard directories. please note that all directory are returned as absolute path with slash "/" between two level of directories. and that it always ends with "\". e.g. "c:/lxzsrc/paraengineSDK/" or "c:/lxzsrc/paraengineSDK/script/"
Return : the directory is returned.
Parameters
- `dwDirectoryType`: it can be one of the PARAENGINE_DIRECTORY enumeration type enum PARAENGINE_DIRECTORY{ APP_ROOT_DIR=0, APP_SCRIPT_DIR=1, APP_ARCHIVE_DIR=2, // xmodels APP_MODEL_DIR=3, APP_SHADER_DIR=4, APP_DATABASE_DIR=5, APP_TEMP_DIR=6, APP_USER_DIR=7, APP_BACKUP_DIR=8, APP_SCREENSHOT_DIR=9, APP_PLUGIN_DIR=10, };

const char ***`GetCurDirectory_`**(DWORD *dwDirectoryType*)
this should never be called from the scripting interface. it is only for API exportation. it uses a static string for the output. so it is not thread-safe.

string **`GetParentDirectoryFromPath`**(**const** char **sfilename*, int *nParentCounts*)
trim the sFile by nParentCounts number of parent directories.
Return : return "" if the input directory does not have that many parent directories. e.g. "C:/abc/" = GetDirectoryFromPath("C:/abc/aaa",0); "C:/" = GetDirectoryFromPath("C:/abc/",1);
Parameters
- `nParentCounts`: number of parent directory to remove

const char ***`GetParentDirectoryFromPath_`**(**const** char **sfilename*, int *nParentCounts*)
this should never be called from the scripting interface. it is only for API exportation. it uses a static string for the output. so it is not thread-safe.

string **`AutoFindParaEngineRootPath`**(**const** char **sFile*)
This will find the root path from a given directory path using the following rule: find a file called "ParaEngine.sig" in the parent directories of sFile, from near to far. e.g. if sFile is "c:/a/b/c/xxx.x", then it will search for "c:/a/b/c/","c:/a/b/","c:/a/" and "c:/". the function will return the first parent directory that contains the file, otherwise "" is returned.

const char ***`AutoFindParaEngineRootPath_`**(**const** char **sFile*)
this should never be called from the scripting interface. it is only for API exportation. it uses a static string for the output. so it is not thread-safe.

string **`ChangeFileExtension`**(**const** char **sFile*, **const** string &*sExt*)
change the file extension.
Return : return the file with the changed extension. the input file does not contain a valid file extension, the returned string will be identical to the input file.
Parameters
- `sFile`: the file whose extension to change.
- `sExt`: the file extension to change to. such as "dds","x"

const char ***`ChangeFileExtension_`**(**const** char **sFile*, **const** string &*sExt*)
this should never be called from the scripting interface. it is only for API exportation. it uses a static string for the output. so it is not thread-safe.

string **`GetFileExtension`**(**const** char **sFile*)
get the file extension. this function may return "" if no file extension is found

const char ***`GetFileExtension_`**(**const** char **sFile*)
this should never be called from the scripting interface. it is only for API exportation. it uses a static string for the output. so it is not thread-safe.

string **`GetRelativePath`** (**const** char **sAbsolutePath*, **const** char **sRootPath*)

Get the relative file path by stripping the root path from the beginning. please note that all paths should uses slash "/", instead of backslash "\", in the path name.letter case is ignored

Return : the relative path is returned. If the absolute path does not math the root path, the absolute path is returned unchanged. the relative path does not begin with "/" e.g. "a/b.x" = GetRelativePath("c:/lxzsrc/a/b.x", "c:/lxzsrc/"); "c:/lxzsrc/a/b.x" = GetRelativePath("c:/lxzsrc/a/b.x", "c:/srclxz/"); // not match

Parameters

- `sAbsolutePath`: the absolute path from which to obtain the relative path.
- `sRootPath`: the parent root path, which will be removed from the absolute path. It should end with "/"

const char ***`GetRelativePath_`** (**const** char **sAbsolutePath*, **const** char **sRootPath*)

this should never be called from the scripting interface. it is only for API exportation. it uses a static string for the output. so it is not thread-safe.

string **`GetAbsolutePath`** (**const** char **sRelativePath*, **const** char **sRootPath*)

Get the absolute file path by appending the root path before the relative path. please note that all paths should uses slash "/", instead of backslash "\", in the path name. letter case is ignored

Return : the relative path is returned. If the absolute path does not math the root path, the absolute path is returned unchanged. e.g. "c:/lxzsrc/a/b.x" = GetAbsolutePath("a/b.x", "c:/lxzsrc/");

Parameters

- `sRelativePath`: the absolute path from which to obtain the relative path. It should not begin with "/"
- `sRootPath`: the parent root path, which will be removed from the absolute path. It should end with "/"

const char ***`GetAbsolutePath_`** (**const** char **sRelativePath*, **const** char **sRootPath*)

this should never be called from the scripting interface. it is only for API exportation. it uses a static string for the output. so it is not thread-safe.

string **`GetFileName`** (**const** char **sFilePath*)

get only the file name from the file path. "a.x" = GetFileName("c:/lxzsrc/a.x");

const char ***`GetFileName_`** (**const** char **sFilePath*)

this should never be called from the scripting interface. it is only for API exportation. it uses a static string for the output. so it is not thread-safe.

int **`GetFileSize`** (**const** char **sFilePath*)

The GetFileSize function retrieves the size of the specified file. The file size that can be reported by this function is limited to a DWORD value

Return : size of the file. If the file does not exist or the file size is 0, the return value is 0.

Note : only disk file is searched.files inside zip file are ignored.

bool **`GetFileInfo`** (**const** char **sFilePath*, **const** object &*inout*)

get file info

Parameters

- `inout`: {modification, attr, access, create, size, mode="file|directoy|fileinzip|", fullpath=string}

string **`ToCanonicalFilePath`** (**const** char **sfilename*, bool *bBackSlash*)

convert a file name to canonical file path

Return : the canonical file path name returned.

Parameters

- `sfilename`: it is assumed that strlen(filename) <= MAX_PATH

- `bBackSlash`: if true, the file will use '\'; otherwise use '/'. '\' is win32 compatible. '/' is more user friendly.

const char ***`ToCanonicalFilePath__`**(**const** char **sfilename*, bool *bBackSlash*)
this should never be called from the scripting interface. it is only for API exportation. it uses a static string for the output. so it is not thread-safe.

void **`SetDiskFilePriority`**(int *nPriority*)
set the disk file priority. it affects whether the disk will be searched first or the one in the archive files. default disk file will be searched first.
Parameters
- `nPriority`: 0 is the same priority as the disk file. so 0 or above will cause the disk file to be searched before archive files. below 0, such as -1 will cause the archive files go first.

int **`GetDiskFilePriority`**()
set the disk file priority. it affects whether the disk will be searched first or the one in the archive files. default disk file will be searched first.
Return : 0 is the same priority as the disk file. so 0 or above will cause the disk file to be searched before archive files. below 0, such as -1 will cause the archive files go first.

ParaScripting::*ParaFileSystemWatcher* **`GetFileSystemWatcher`**(**const** char **filename*)
create and get a file system watcher object. always use local to retrieve the object to ensure that the object is properly released when out of scope.

void **`DeleteFileSystemWatcher`**(**const** char **name*)
delete a watcher, it will no longer receive callbacks. note that if someone else still keeps a pointer to the directory watcher, it will not be deleted.

class **`ParaMiniSceneGraph`**
#include <ParaScriptingScene.h> *ParaMiniSceneGraph* class:

Public Functions

const char ***`GetName`**()
get name

void **`SetName`**(**const** char **sName*)
set the object name. this function can be used to rename this object

bool **`IsValid`**() **const**
check if the object is valid

ParaAttributeObject **`GetAttributeObject`**()
get the attribute object associated with the global scene.

ParaScripting::*ParaAttributeObject* **`GetAttributeObject1`**(**const** char **name*)
get the attribute object associated with the global scene.
Parameters
- `name`: "" for global scene, "sky" for sky, "camera" for camera, "sun" for sun.

void **`GetAttributeObject_`**(*ParaAttributeObject* &*output*)
used for API exportation.

ParaScripting::*ParaAttributeObject* **`GetAttributeObjectCamera`**()
get the attribute object associated with the current camera object.

bool **CreateSkyBox**(**const** char **strObjectName*, **const** char **strMeshAssetName*, float *fScaleX*, float *fScaleY*, float *fScaleZ*, float *fHeightOffset*)

create a sky box and add it to the current list. sky box with the same name will not be recreated,but will be selected as the current sky box. It may be a sky box/dome/plane or whatever. The associated mesh will be scaled by the specified amount along x,y,z axis and then translate up or down along the y axis. in many cases, the mesh data in the mesh asset is of unit size.

Parameters

- `strObjectName`: sky name
- `strMeshAssetName`: mesh asset name. this is not the file name.
- `fScaleX`: the static mesh local transform scale along the x axis
- `fScaleY`: the static mesh local transform scale along the y axis
- `fScaleZ`: the static mesh local transform scale along the z axis
- `fHeightOffset`: the translation along the y axis.

void **DeleteSkyBox**(**const** char **strObjectName*)

delete a name sky box.

Parameters

- `strObjectName`: if this is “”, all sky boxes will be deleted.

void **EnableLighting**(bool *bEnable*)

Enable both global and local lighting. Turn off lighting will greatly improve performance, such as on slower computers

void **SetTimeOfDaySTD**(float *time*)

set standard time. see SetTimeOfDay()

Parameters

- `time`: always in the range [-1,1], 0 means at noon, -1 is morning. 1 is night.

float **GetTimeOfDaySTD**()

get standard time. see GetTimeOfDay()

Return : always in the range [-1,1], 0 means at noon, -1 is morning. 1 is night.

void **SetFog**(bool *bRenderFog*, **const** char **strFogColor*, float *fFogStart*, float *fFogEnd*, float *fFogDensity*)

set the global fog effect

Parameters

- `bRenderFog`: 1 to enable fog.
- `strFogColor`: a string of RGB value in the format “%f %f %f”, such as “1.0 1.0 1.0”, value must be in the range [0, 1.0].
- `fFogDensity`: between (0,1)
- `fFogStart`: unit in meters.
- `fFogEnd`: unit in meters.

bool **IsVisible**()

invisible object will not be drawn. e.g. one can turn off the visibility of physics object.

void **SetVisible**(bool *bVisible*)

set the visibility of this object. The visibility will recursively affect all its child objects.

void **EnableCamera**(bool *bEnable*)

enable or disable a given camera

bool **IsCameraEnabled**()

whether camera is enabled. it is disabled by default. it is much faster to disable camera, because it will use the main scene’s render pipeline and effects. Otherwise it will be rendered after the main scene is rendered, since the camera is different. TODO: currently mini scene graph is only rendered when its camera is disabled. local camera is not supported at the moment

ParaScripting::*ParaObject* **`GetObject`** (**const** char **name*)
update the camera parameter by providing the lookat and eye position get the camera parameter of the lookat and eye position in scripting interface all inputs are outputs. get object by name, if there are multiple objects with the same name, the last added one is inserted.

Note : This function will traverse the scene to search the object. So there might be some performance penalty.

Parameters
- `name`:

ParaObject **`GetObject3`** (float *x*, float *y*, float *z*)
get the first object that matches the position. EPSILON is 0.01f

ParaObject **`GetObject4`** (float *x*, float *y*, float *z*, float *fEpsilon*)
get the first object that matches the position within fEpsilon, which is usually 0.01f

int **`RemoveObject`** (**const** char **name*)
remove an object from this scene graph but do not destroy it. this function can be used to move a node from one scene graph to another

int **`RemoveObject_`** (**const** *ParaObject* &*pObj*)
remove an object from this scene graph but do not destroy it. this function can be used to move a node from one scene graph to another

void **`AddChild`** (**const** *ParaObject* *obj*)
attach the object as its child object.

Parameters
- `obj`: the child object to attach

int **`DestroyObject`** (**const** char **name*)
destroy all objects with the given name. the current version will only destroy the first met child with the given name.

Return the number of objects deleted.

int **`DestroyObject_`** (**const** *ParaObject* &*pObj*)
delete an object from this scene graph but do not destroy it.This function will search the scene recursively this function can be used to move a node from one scene graph to another Note: this is like calling RemoveObject(pObj) and then delete the object.

Parameters
- `pObj`: object to delete.

void **`DestroyChildren`** ()
destroy all children of this mini-scenegraph. but still preserving other settings like camera and render target.

void **`Reset`** ()
clear the entire scene graph

void **`SetActor`** (**const** *ParaObject* *pActor*)
set the actor: The camera always focuses on actor, so this actor can be used to control the current camera position.

Parameters
- `pActor`: it must be a valid object.

ParaScripting::*ParaObject* **`GetActor`** ()
get the current actor

ParaScripting::*ParaAssetObject* **GetTexture**()

the canvas texture, which can be used as any other ordinary texture on 3D or 2D object.

Return

void **CameraZoom**(float *fAmount*)

Zoom the camera

Parameters

- `fAmount`:

void **CameraZoomSphere**(float *center_x*, float *center_y*, float *center_z*, float *raduis*)

reset the camera parameters to view the entire sphere at best (default) distance

Parameters

- `center_x`: sphere center x
- `center_y`: sphere center y
- `center_z`: sphere center z
- `raduis`: sphere raduis

void **CameraRotate**(float *dx*, float *dy*, float *dz*)

rotate the camera round the object on canvas

Parameters

- `dx`:
- `dy`: relative amount in radian.
- `dz`:

void **CameraPan**(float *dx*, float *dy*)

pan the camera

Parameters

- `dx`: relative amount in pixels
- `dy`: relative amount in pixels

void **CameraSetLookAtPos**(float *x*, float *y*, float *z*)

set the camera look at position

void **CameraSetEyePosByAngle**(float *fRotY*, float *fLiftupAngle*, float *fCameraObjectDist*)

set the camera eye position

Parameters

- `fRotY`: rotation of the camera around the Y axis, in the world coordinate.
- `fLiftupAngle`: lift up angle of the camera.
- `fCameraObjectDist`: the distance from the camera eye to the object being followed.

void **Draw**(float *fDeltaTime*)

draw the content of the scene graph to the current render target. If EnableActiveRendering is enabled, this function will be called each render frame after the main scene graph. however, if EnableActiveRendering is disabled, one can call this function to render a single frame to the render target on demand.

void **SaveToFile**(**const** char **sFileName*, int *nImageSize* = 0)

save to file.

Parameters

- `sFileName`: a texture file path to save the file to. we support ".dds", ".jpg", ".png" files. If the file extension is not recognized, ".png" file is used.
- `nImageSize`: if this is zero, the original size is used. If it is dds, all mip map levels are saved.

void **SetRenderTargetSize**(int *nWidth*, int *nHeight*)

set the canvas size in pixels

Parameters

- `nWidth`: default to 512
- `nHeight`: default to 512

void **`SetBackGroundColor`**(**const** char **rgba*)

set the color of the scene ground when it is not enabled.When scene is enabled, the background color is always the fog color.

Parameters

- `rgba`: it can contain alpha channel, such as "255 255 255 128". If no alpha is specified like "255 255 255", alpha will be 1.0

void **`SetMaskTexture`**(*ParaAssetObject pTexture*)

this is an optional 2D mask, which is drawn over the entire canvas after scene is rendered in to it.

Parameters

- `pTexture`:

void **`EnableActiveRendering`**(bool *bEnable*)

if true, contents will be drawn to the current render target each frame. Default value is false. where the user can call DrawToTexture() on demand.

bool **`IsActiveRenderingEnabled`**()

if true, contents will be drawn to the current render target each frame. Default value is false. where the user can call DrawToTexture() on demand.

ParaScripting::*ParaObject* **`MousePick`**(float *x*, float *y*, float *fMaxDistance*, **const** char **sFilterFunc*)

Pick scene object at the current mouse cursor position. pick the smallest intersected object which is un-occluded by any objects. Object A is considered occluded by object B only if (1) both A and B intersect with the hit ray. (2) both A and B do not intersect with each other. (3) B is in front of A, with regard to the ray origin.

this function will ray-pick any loaded scene object(biped & mesh, but excluding the terrain) using their oriented bounding box. a filter function may be provided to further filter selected object. this function will transform all objects to near-camera coordinate system. This will remove some floating point inaccuracy near the camera position.Hence this function is most suitable for testing object near the camera eye position. This function does not rely on the physics engine to perform ray-picking.

Return :the scene object. if the object is invalid, it means that the ray has hit nothing.

Parameters

- `x`: screen position relative to the render target.
- `y`: screen position relative to the render target. fMaxDistance: the longest distance from the ray origin to check for collision. If the value is 0 or negative, the view culling radius is used as the fMaxDistance.
- `sFnctFilter`: it can be any of the following string or a number string "mesh": mesh any mesh object in the scene. Usually for selection during scene editing. "cmesh": mesh object that is clickable (associated with scripts). Usually for game playing. "notplayer": any object in the scene except for the current player. Usually for selection during scene editing. "": any object in the scene except. Usually for selection during scene editing. "light": only pick light objects "biped": any character objects :local or global. "anyobject": any objects, including mesh and characters. but not including helper objects, such as light. "global": all global objects, such as global character and mesh. This is usually for game mode. "point": the returned object is invalid if there no collision with any physics faces. otherwise, one can use GetPosition function of the returned object to retrieve the intersection point. "terrain": pick a point on the global terrain only. "walkpoint": pick a point on the global terrain or any physical object with camera obstruction attribute set to true. "actionmesh": mesh with action script. number: if it is a number, it is treated as a 32 bitwise DWORD filter code. see SetPickingFilter() for more example.

void **`ShowHeadOnDisplay`** (bool *bShow*)
show or hide all scene's objects' head on display

bool **`IsHeadOnDisplayShown`** ()
whether all scene's objects' head on display

class `ParaMisc`
#include <ParaScriptingMisc.h> Contains miscellaneous functions

Public Static Functions

int **`GetUnicodeCharNum`** (**const** char **str*)
get the number of characters in str. Str is assumed to be in ANSI code page. it is converted to Unicode and return the character count.

std::string **`UniSubString`** (**const** char **str*, int *nFrom*, int *nTo*)
same as LUA string.sub(), except that the index is character. get a sub string of a ANSI Code page string. However, the index are unicode characters.
Parameters

- `str`: the string to use
- `nFrom`: character index beginning from 1.

const char ***`SimpleEncode`** (**const** char **source*)
encode a string using really simple algorithm. it just makes the source ineligible. It is still not immune to crackers. str = SimpleDecode(SimpleEncode(str))
Return : it may return NULL if input invalid

const char ***`SimpleDecode`** (**const** char **source*)
decode a string using really simple algorithm. str = SimpleDecode(SimpleEncode(str))
Return : it may return NULL if input invalid

string **`md5`** (**const** char **source*)
convert the md5 of the input source string.

long **`RandomLong`** (**const** object *&seedTable*)
Generating [-MAX, MAX] long integer
Parameters

- `seedTable`: nil or a table containing {_seed=integer} when the function returned the seedTable._seed will be modified.

double **`RandomDouble`** (**const** object *&seedTable*)
generating [0,1] double value

const char ***`EncodingConvert`** (**const** object *&srcEncoding*, **const** object *&dstEncoding*, **const** object *&bytes*)
Converts an entire byte array from one encoding to another.
Parameters

- `srcEncoding`: any encoding name. If nil or "", it is the default coding in NPL. see Encoding.GetEncoding(). Below are some commonly used field | *Code Page* | *Name* | | 950 | big5 | | 936 | gb2312 | | 65001 | utf-8 | | 65005 | utf-32 | There is one special code name called "HTML", which contains HTML special characters in ascii code page. This is usually true for most "iso8859-15" encoding in western worlds. It just writes the unicode number+XXXX in ascii character "&#XXXX;" where & is optional.

- `dstEncoding`: save as above. If nil or "", it will be converted to default coding in NPL. : the source bytes. e.g. The most common use of this function is to create HTML special character to NPL string, like below local text = *ParaMisc.EncodingConvert*("HTML", "", "Chinese characters: &#24320;&#21457;") log(text);

bool **`CopyTextToClipboard`**(**const** char **text*)
copy text to clipboard. Input is ANSI code page

const char ***`GetTextFromClipboard`**()
get text from clipboard. text is converted to ANSI code page when returned.

class `ParaMovie`
#include <ParaScriptingMovie.h> movie making and screen capture functions

Public Static Functions

ParaScripting::*ParaAttributeObject* **`GetAttributeObject`**()
get the attribute object associated with an object.

void **`SetMovieScreenSize`**(int *nWidth*, int *nHeight*)
set the movie screen size in pixel.
Parameters
- `nWidth`: in pixels, must be multiple of 4.
- `nHeight`: in pixels, must be multiple of 4.

void **`SetCaptureGUI`**(bool *bGUI*)
set whether GUI is captured.

bool **`CaptureGUI`**()
return true if GUI is also captured.

void **`SetRecordingFPS`**(int *nFPS*)
set the recording FPS, the default is 20 FPS. Some may prefer 30FPS.

int **`GetRecordingFPS`**()
Get the recording FPS, the default is 20 FPS. Some may prefer 30FPS.

void **`SetStereoCaptureMode`**(int *nMode*)
set the stereo capture mode. This is used to generate video files that can be viewed by 3d eye glasses and stereo video player.
- 0 for disable stereo capture(default);
- 1 for line interlaced stereo.
- 2 for left right stereo;
- 3 for above below stereo;
- 4 for frame interlaved mode, where the odd frame is the left eye and even frame is the right image;

int **`GetStereoCaptureMode`**()
Get the stereo capture mode. This is used to generate video files that can be viewed by 3d eye glasses and stereo video player.
- 0 for disable stereo capture(default);
- 1 for iPod mode, where the odd frame is the left eye and even frame is the right image;
- 2 for left right stereo;
- 3 for above below stereo;
- 4 for interlaced stereo.

void **SetStereoEyeSeparation** (float *fDist*)
the distance in meters between the left and right eye when generating the stereo image. some common values are in range [0.03, 0.1]. This is also related to the rendering unit that we used in games. since ParaEngine games usually use meter as its rendering unit, the value is such near the real eye separation distance.

float **GetStereoEyeSeparation** ()
the distance in meters between the left and right eye when generating the stereo image. some common values are in range [0.03, 0.1]. This is also related to the rendering unit that we used in games. since ParaEngine games usually use meter as its rendering unit, the value is such near the real eye separation distance.

bool **BeginCapture** (string *sFileName*)
starting capturing a screen movie
Parameters
- `sFileName`: the movie file name, which can be "". If it is "", a default name is used.

bool **EndCapture** ()
end capturing a screen movie and save movie to file.

void **PauseCapture** ()
pause the capturing.

void **ResumeCapture** ()
resume capturing

bool **IsInCaptureSession** ()
whether we are doing screen capture. I.e. true between *BeginCapture()* and *EndCapture()* However, it may be in the recording or paused state.

bool **IsRecording** ()
whether it is recording.

bool **FrameCapture** ()
capture a given frame

void **GetMovieScreenSize** (int **nWidth*, int **nHeight*)
get movie screen size. In script. Use like this : local x,y = *ParaMovie.GetMovieScreenSize()*;

string **GetMovieFileName** ()
get the movie file name

void **SelectCodecOptions** ()
display a dialog which allows the user to set the codec to be used.

void **SetEncodeMethod** (int *eType*)
set the code
Parameters
- `eType`: 0 for XVID; 1 for WMV; -1 for user select codec.

int **GetEncodeMethod** ()
See *SetEncodeMethod()*

bool **TakeScreenShot** (**const** char **filename*)
we will automatically take screen shot according to the file extensions, supported file extensions are "jpg","dds","bmp","tga",
Note : "jpg" has small file size; where "bmp" and "tga" is lossless.
Parameters

- `filename`: this is the file name.If this is NULL or "", it will be automatically named as jpg file under the "screen shots" directory.

bool **`TakeScreenShot3`**(**const** char **filename*, int *width*, int *height*)

render the current scene to texture, UI are disabled by default. Aspect ratio are changed according to width/height. supported file extensions are "jpg","dds","bmp","tga",

Note : "jpg" has small file size; where "bmp" and "tga" is lossless.

Parameters

- `filename`: this is the file name.If this is NULL or "", it will be automatically named as jpg file under the "screen shots" directory.
- `width;`: in pixel, if 0 it will be the screen size
- `height;`: in pixel, if 0 it will be the screen size

bool **`ResizeImage`**(**const** string &*filename*, int *width*, int *height*, **const** char **destFilename*)

resize the given image. It can also be used to change the file format

Parameters

- `filename`: source file name
- `width;`: in pixel
- `height;`: in pixel
- `destFilename`: destination file name. If nil or "", it will be the same as input. It can also be used to change the file format

void **`GetImageInfo`**(**const** string &*filename*, int **width*, int **height*, int **nFileSize*)

get the given image info: i.e. size e.g. local width, height, filesize = *ParaMovie.GetImageInfo*("abc.jpg")

Parameters

- `width;`: out in pixel
- `height;`: out in pixel
- `nFileSize`: out in size in bytes

class **`ParaMovieCtrler`**

#include <ParaScriptingCharacter.h> *ParaMovieCtrler* object: it is used to control time based movie of character object.

Public Functions

bool **`IsValid`**()

check if the object is valid

string **`GetTime`**(**const** char **sFormat*)

get the local time of a module of the character

Parameters

- `sFormat`: the format of the time in the returning string "": in seconds Formated: The format argument consists of one or more codes; as in printf, the formatting codes are preceded by a percent sign (%). Characters that do not begin with % are copied unchanged. H : Hour in 24-hour format (00 C 23) M : Minute as decimal number (00 C 59) S : Second as decimal number (00 C 59) e.g.: GetTime("movie", "time elapsed: %H:%M:%S");

void **`SetTime`**(float *fTime*)

set the current time of the movie

Parameters

- `fTime`: in seconds. 0 is the beginning of the movie

void **`Suspend`**()

suspend the movie

void **Resume**()
: resume the movie

void **Play**()
: play the movie

void **Record**()
: record the movie from its current position. all movies behind will be erased.

void **RecordNewDialog**(**const** char **sDialog*)
: add a new dialog record to the current time. All trailing dialog will be removed.

bool **RecordNewAction**(**const** char **sActionName*)
: add a new action record to the current time. All trailing dialog will be removed.

bool **RecordNewAction_**(**const** char **sActionName*, float *fTime*)
: add a new action record to the current time. All trailing dialog will be removed.

void **RecordNewEffect**(int *effectID*, **const** char **sTarget*)
: add a new effect record to the current time. All trailing dialog will be removed.

void **GetOffsetPosition**(float **x*, float **y*, float **z*)
: get the offset position of the movie. All movie's position key are relative to it. if there is no position key, 0,0,0 is returned e.g. x,y,z = movieCtrler:*GetOffsetPosition()*;

void **SetOffsetPosition**(float *x*, float *y*, float *z*)
: set the offset position of the movie. All movie's position key are relative to it.

void **SaveMovie**(**const** char **filename*)
: save recorded movie to file.

  Parameters
  - `filename:file`: path to save the movie

void **LoadMovie**(**const** char **filename*)
: load a recorded movie from file.

  Parameters
  - `filename:file`: path to load the movie

class **ParaNetwork**
: *#include <ParaScriptingNetwork.h>* API wrapper for NPL Network Layer functions

Public Static Functions

void **EnableNetwork**(bool *bEnable*, **const** char **CenterName*, **const** char **password*)
: Enable the network, by default the network layer is disabled. calling this function multiple time with different CenterName will restart the network layer with a different center name.

  Return true if succeeded.

  Parameters
  - `bEnable`: true to enable, false to disable.If this is false, the CenterName and Password are ignored.
  - `CenterName`: the local nerve center name. it is also the user name which local receptor will use in the credentials to login in other NPL runtime.
  - `Password`:

bool **IsNetworkLayerRunning**()
: whether network layer is running or not.

Return

void **`SetNerveCenterAddress`**(**const** char **pAddress*)
set the IP and port for the nerve center: call this function before you enable the network (EnableNetwork) for it to take effect.
Parameters
- `pAddress`: such as "127.0.0.1:60001", or ":60001" (if you want to use any available IP on the computer)

void **`SetNerveReceptorAddress`**(**const** char **pAddress*)
set the IP and port for the nerve receptor: call this function before you enable the network (EnableNetwork) for it to take effect.
Parameters
- `pAddress`: such as "127.0.0.1:60000", or ":60000" (if you want to use any available IP on the computer)

class `ParaNPLRuntimeState`
#include <ParaScriptingNPL.h> A runtime state contains the scripting runtime stack and can be run in a single thread.

An NPL runtime state is message driven, however we also support timer and several other event callbacks. Normally we only use this class to start a new runtime, or get statistics about a runtime

Public Functions

bool **`IsValid`**()
if this is a valid state.

const char ***`GetName`**() **const**
return the name of this runtime state. if "", it is considered an anonymous name

int **`Start`**()
start this runtime state in a worker thread
Return the number of threads that are currently working on the runtime state. normally this is 1. only dll runtime state may have more than one worker thread.

bool **`Stop`**()
Stop the worker thread. this will stop processing messages in the thread.

void **`Reset`**()
it is like starting a fresh new runtime state. All memory, tables, timers, pending messages are removed. this function only takes effect on the next message loop. So there can be other clean up code following this function.
Parameters
- `onResetScode`: the code to be executed immediately after runtime state is reset. default to NULL.

luabind::object **`GetStats`**(**const** object &*inout*)
TODO: get statistics about this runtime environment.

int **`GetCurrentQueueSize`**()
Get the current number of messages in the input message queue. Sometimes, a monitor or dispatcher logics may need to know the queue size of all NPL runtime states. and a dispatcher will usually need to add new messages to the NPL state with smallest queue size. This function will lock the message queue to retrieve the queue size, so do not call it very often, but use a timer to query it on some interval. [thread safe]

int **`GetProcessedMsgCount`** ()
: the total number of message processed by this runtime state since start. If this number does not increase, perhaps the processing is blocking somewhere, and we should take actions. [Not thread safe] in most cases, it will return correct result even in multi-threaded environment, but since we do not use lock, unexpected result may return. This function is usually used for stat printing and monitoring.

int **`GetMsgQueueSize`** ()
: get the message queue size. default to 500. For busy server side, we can set this to something like 5000 [thread safe]

void **`SetMsgQueueSize`** (int *nSize* = 500)
: set the message queue size. default to 500. For busy server side, we can set this to something like 5000 [thread safe]

void **`WaitForMessage`** ()
: simply wait for the next message to arrive.

void **`WaitForMessage2`** (int *nMessageCount*)
: **Parameters**
    - `nMessageCount`: if not negative, this function will immediately return when the message queue size is bigger than this value.

luabind::object **`PeekMessage`** (int *nIndex*, **const** object &*inout*)
: **Return** : msg table {filename, code, msg} if exist, or nil.
  Parameters
    - `inout`: this should be a table {filename=true, code=true, msg=true}, specify which part of the message to retrieve in return value. {filename=true} will only retrieve the filename, because it is faster if code is big.

luabind::object **`PopMessageAt`** (int *nIndex*, **const** object &*inout*)
: pop message at given index. usually we need to call peek() first.
  Return : msg table {filename, code, msg, result} if exist, or filename will be false. result is only available if process is true
  Parameters
    - `inout`: this should be a table {filename=true, code=true, process=true}, specify which part of the message to retrieve in return value. {filename=true} will only retrieve the filename, because it is faster if code is big. if inout.process == true, we will pop and process the message via standard activation. if inout.process == false, we will pop without processing the message, in which case the caller may need to process it manually

class **`ParaObject`**
: *#include <ParaScriptingScene.h>* *ParaObject* class: it is used to control game scene objects from scripts.

  •("name",&*ParaObject::GetName*,&*ParaObject::SetName*)

  •("onclick",&*ParaObject::GetOnClick*,&ParaObject::SetOnClick)

  •("onentersentientarea",&*ParaObject::GetOnEnterSentientArea*,&ParaObject::SetOnEnterSentientArea)

  •("onleavesentientarea",&*ParaObject::GetOnLeaveSentientArea*,&ParaObject::SetOnLeaveSentientArea)

  •("onperceived",&*ParaObject::GetOnPerceived*,&ParaObject::SetOnPerceived)

  •("onframemove",&*ParaObject::GetOnFrameMove*,&ParaObject::SetOnFrameMove)

  •("onnetreceive",&*ParaObject::GetOnNetReceive*,&ParaObject::SetOnNetReceive)

  •("onnetsend",&*ParaObject::GetOnNetSend*,&ParaObject::SetOnNetSend)

Class Properties

Public Functions

const char ***GetType**()
get the Runtime class information of the object.

int **GetMyType**() **const**
get paraengine defined object type name. TODO: This function is not implemented

int **GetID**()
the ID of the object. The ID of the object is only generated when the first time this function is called. One can then easily get the object, by calling ParaScene.GetObject(nID). When an object is released, its ID is not longer used. Please note that we can ParaScene.GetObject(nID) even if the object is never attached before. Please note that the ID is neither unique outside the world, nor persistent in the same world.
Return : return 0 if object is invalid.

bool **IsValid**() **const**
check if the object is valid

bool **IsAttached**() **const**
whether the object has been attached to the scene.

ParaAssetObject **GetPrimaryAsset**()
get the main asset object associated with this object. the object may be invalid.

void **GetPrimaryAsset_**(*ParaAssetObject* **pOut*)
this function shall never be called from the scripting interface. this is solely for exporting API. and should not be used from the scripting interface.

ParaScripting::*ParaParamBlock* **GetEffectParamBlock**()
get the parameter block of the effect (shader) associated with this object.

bool **IsPersistent**()
whether the object is persistent in the world. If an object is persistent, it will be saved to the world's database. if it is not persistent it will not be saved when the world closes. Player, OPC, some temporary movie actors may by non-persistent; whereas NPC are usually persistent to the world that it belongs.

void **SetPersistent**(bool *bPersistent*)
See *IsPersistent()*

bool **SaveToDB**()
save the current character to database according to the persistent property. if the object is persistent, the action is to update or insert the character to db if the object is non-persistent, the action is to delete it from the database.

bool **equals**(**const** *ParaObject* *obj*) **const**
return true, if this object is the same as the given object.

const char ***ToString**() **const**
convert the object to object creation string.
Return : "" if not valid

const char ***ToString1**(**const** char **sMethod*) **const**

convert the object to string.

Return : "" if not valid

Parameters

- `sMethod`: it can be one of the following strings "create": generate script to create the object "update": generate script to update the object, useful for updating static physics object "delete": generate script to delete the object "loader": generate script to create the object in managed loader

ParaAttributeObject **GetAttributeObject**()

get the attribute object associated with an object.

void **GetAttributeObject_**(*ParaAttributeObject* &*output*)

for API exportation

void **CheckLoadPhysics**()

when this function is called, it ensures that the physics object around this object is properly loaded. It increases the hit count of these physics objects by 1. The garbage collector in the physics world may use the hit count to move out unused static physics object from the physics scene (Novodex). This function might be called for the current player, each active mobile object in the scene and the camera eye position. vCenter: the center of the object in world coordinates fRadius: the radius of the object within which all physics object must be active.

void **LoadPhysics**()

only load physics of this object if it has not loaded.

luabind::object **GetField**(**const** char **sFieldname*, **const** object &*output*)

get field by name. e.g. suppose att is the attribute object. local bGloble = att:GetField("global", true); local facing = att:GetField("facing", 0); local pos = att:GetField("position", {0,0,0}); pos[1] = pos[1]+100;pos[2] = 0;pos[3] = 10;

Return : return the field result. If field not found, output will be returned. if field type is vectorN, return a table with N items.Please note table index start from 1

Parameters

- `sFieldname`: field name
- `output`: default value. if field type is vectorN, output is a table with N items.

void **SetField**(**const** char **sFieldname*, **const** object &*input*)

set field by name e.g. suppose att is the attribute object. att:SetField("facing", 3.14); att:SetField("position", {100,0,0});

Parameters

- `sFieldname`: field name
- `input`: input value. if field type is vectorN, input is a table with N items.

void **CallField**(**const** char **sFieldname*)

call field by name. This function is only valid when The field type is void. It simply calls the function associated with the field name.

luabind::object **GetDynamicField**(**const** char **sFieldname*, **const** object &*output*)

get field by name. e.g. suppose att is the attribute object. local bGloble = att:GetField("URL", nil); local facing = att:GetField("Title", "default one");

Return : return the field result. If field not found, output will be returned. if field type is vectorN, return a table with N items.Please note table index start from 1

Parameters

- `sFieldname`: field name
- `output`: default value. if field type is vectorN, output is a table with N items.

void **SetDynamicField**(**const** char **sFieldname*, **const** object &*input*)
set field by name e.g. suppose att is the attribute object. att:SetDynamicField("URL", 3.14); att:SetDynamicField("Title", {100,0,0});
Parameters
- `sFieldname`: field name
- `input`: input value. can be value or string type

ParaScripting::*ParaObject* **GetObject**(**const** char **name*)
get object by name, if there are multiple objects with the same name, the last added one is inserted.
Note : This function will traverse the scene to search the object. So there might be some performance penalty.
Parameters
- `name`:

bool **IsStanding**()
whether the object has 0 speed.

bool **IsVisible**()
invisible object will not be drawn. e.g. one can turn off the visibility of physics object.

void **SetVisible**(bool *bVisible*)
set the visibility of this object. The visibility will recursively affect all its child objects.

bool **CheckAttribute**(DWORD *attribute*)
whether an object attribute is enabled.
Return true if enabled
Parameters
- `attribute`: object volume bit fields enum OBJECT_ATTRIBUTE { / two solid objects with sensor volume will cause environment simulator to / to generate sensor/collision event when they come in to contact. OBJ_VOLUMN_SENSOR = 1, / all child objects are in this object's volume OBJ_VOLUMN_CONTAINER = 0x1<<1, / solid objects(like biped) can be placed on its volume, provided / it's not already occupied by any solid objects from its children / when we solve two solid object collision, this is the field we check first. OBJ_VOLUMN_FREESPACE = 0x1<<2, / whether the object is isolated from its siblings. An isolated object / can overlap in physical space with all its siblings regardless of their solidity. / multiple scenes or terrains can be declared as ISOLATED object. Note, the object / is not isolated from its parent, though. OBJ_VOLUMN_ISOLATED = 0x1<<3, / the object has a perceptive radius that may be larger than the object's / collision radius. Currently only biped object might has this volume type OBJ_VOLUMN_PERCEPTIVE_RADIUS = 0x1<<4, / objects with this VIP volume type will trigger the plot of the scene in its view-culling radius. OBJ_VOLUMN_VIP = 0x1<<5, / Object invisible, the object is not drawn.but its physics may load. added by lxz 2006.3.5 OBJ_VOLUMN_INVISIBLE = 0x1<<6, / mask of the above bits. this field is never used externally. VOLUMN_MASK = 0x7f, / whether lights have effects on this object. MESH_USE_LIGHT = 0x1<<7, / whether to rotate the object around Y axis to let the object always facing the camera. MESH_BILLBOARDED= 0x1<<8, / whether it is a shadow receiver. MESH_SHADOW_RECEIVER= 0x1<<9, / whether it is a vegetation. MESH_VEGETATION= 0x1<<10, };

void **SetAttribute**(DWORD *dwAtt*, bool *bTurnOn*)
enable or disable a given attribute.

Below are some attributes. For more information please see BaseObject.h

/ two solid objects with sensor volume will cause environment simulator to / to generate sensor/collision event when they come in to contact. OBJ_VOLUMN_SENSOR = 1, / all child objects are in this object's volume OBJ_VOLUMN_CONTAINER = 0x1<<1, / solid objects(like biped) can be placed on its volume, provided / it's not already occupied by any solid objects from its children / when we solve two solid object collision, this is the field we check first. OBJ_VOLUMN_FREESPACE = 0x1<<2, / whether the object is isolated from its siblings. An isolated object / can overlap in physical space with all its siblings regardless of their solidity. / multiple scenes or terrains can be declared as ISOLATED object. Note, the object / is not isolated from its parent, though. OBJ_VOLUMN_ISOLATED = 0x1<<3, / the object has a perceptive radius that may be larger than the object's / collision radius. Currently only biped object might has this volume type OBJ_VOLUMN_PERCEPTIVE_RADIUS = 0x1<<4, / objects with this VIP volume type will trigger the plot of the scene in its view-culling radius. OBJ_VOLUMN_VIP = 0x1<<5, / Object invisible, the object is not drawn.but its physics may load. added by lxz 2006.3.5 OBJ_VOLUMN_INVISIBLE = 0x1<<6,

Parameters

- `dwAtt`:
- `bTurnOn`: true to turn on, false to turn off.

void **`SetPosition`** (double *x*, double *y*, double *z*)

set world position. Please note, for static object, it may make the quad tree terrain in which the object is located invalid. it may also make the physics engine panic.In such cases, one should call *ParaScene.Attach()* after chancing the position or rotation of a static mesh or physics object. If any of the following rule matches, the function is safe to use.

- Use this function for global biped if (x,y,z) changes.
- Use this function for all objects if it has not been attached to the global terrain
- Use this function for static mesh, with (0,y,0) only.i.e. only changing height.
- Never use this function for physics object, unless you are building the world. If you do use it it with physics or static mesh object, make sure that you follow the following rules: create the physics object and save it in a global variable called physicsObj. physicsObj = ParaScene.Attach(physicsObj); attach the physics object to the scene modification of the position, orientation, scaling of the object occurred. local x,y,z = physicsObj:*GetPosition()*; physicsObj:SetPosition(x+2,y,z); immediately call Attach again with physicsObj, so that the physical scene will be updated and the object be re-inserted properly in to the scene. physicsObj = ParaScene.Attach(physicsObj);

  Parameters

  - `x`: global x
  - `y`: global y
  - `z`: global z

void **`GetPosition`** (double **x*, double **y*, double **z*)

get the world position of the object. This function takes no parameters. x,y,z are not input, but pure output. In the script, we can call it as below x,y,z = biped:*GetPosition()*; get the biped's position in Luabind, it is defined as .def("GetPosition", &*ParaObject::GetPosition*, pure_out_value(_2) + pure_out_value(_3) + pure_out_value(_4)) please note, y is the absolute position in world coordinate

See *SetPosition(double x, double y, double z)*

void **`GetViewCenter`** (double **x*, double **y*, double **z*)

get the world position of the center of the view object. This function takes no parameters. x,y,z are not input, but pure output. In the script, we can call it as below x,y,z = biped:*GetViewCenter()*; get the biped's center

void **`OffsetPosition`** (float *dx*, float *dy*, float *dz*)

offset the current object position by (dx,dy,dz)

See SetPosition(float x, float y, float z) for precautions of using this function

void **`SetFacing`** (float *fFacing*)

set object facing around the Y axis. this function is safe to call for all kind of objects except the physics mesh object. for physics mesh object, one must call *ParaScene.Attach()* immediately after this function. for more information, please see SetPostion();

See : SetPostion();

float **`GetFacing`**()
get object facing around the Y axis

void **`Rotate`**(float *x*, float *y*, float *z*)
Rotate the object.This only takes effects on objects having 3D orientation, such as static mesh and physics mesh. The orientation is computed in the following way: first rotate around x axis, then around y, finally z axis. Note: this function is safe to call for all kind of objects except the physics mesh object. for physics mesh object, one must call *ParaScene.Attach()* immediately after this function. for more information, please see SetPostion();
See : SetPostion();
Parameters

- `x`: rotation around the x axis.
- `y`: rotation around the y axis.
- `z`: rotation around the z axis.

void **`Scale`**(float *s*)
set the scale of the object. This function takes effects on both character object and mesh object. Note: this function is safe to call for all kind of objects except the physics mesh object. for physics mesh object, one must call *ParaScene.Attach()* immediately after this function. for more information, please see SetPostion();
See : SetPostion();
Parameters

- `s`: This is a relative scale to its current size. Scaling applied to all axis.1.0 means original size.

float **`GetScale`**()
set the scale of the object. This function takes effects on both character object and mesh object. Note: this function is safe to call for all kind of objects except the physics mesh object. for physics mesh object, one must call *ParaScene.Attach()* immediately after this function. for more information, please see SetPostion();
Parameters

- `s`: this is the absolute scale on the original mesh model. Scaling applied to all axis.1.0 means original size.

void **`SetScale`**(float *s*)
set scale
See *GetScale()*;
Parameters

- `s`:

object **`GetRotation`**(**const** object *&quat*)
this usually applies only to mesh object. get the rotation as quaternion. e.g. local mat3x3 = obj:GetRotation({});
Return the rotational matrix is of the following format: {x,y,z,w,}

void **`SetRotation`**(**const** object *&quat*)
set the rotation as quaternion.
Parameters

- `sRot`: the rotational matrix is of the following format: {x,y,z,w,}

void **`Reset`**()
reset the object to its default settings.

void **`SetPhysicsGroup`**(int *nGroup*)
set the physics group ID to which this object belongs to default to 0, must be smaller than 32. please see groups Mask used to filter shape objects. See #NxShape::setGroup

- group 0 means physics object that will block the camera and player, such as building walls, big tree trunks, etc.
- group 1 means physics object that will block the player, but not the camera, such as small stones, thin posts, trees, etc.

int **GetPhysicsGroup**()

Get the physics group ID to which this object belongs to default to 0, must be smaller than 32. please see groups Mask used to filter shape objects. See #NxShape::setGroup

- group 0 means physics object that will block the camera and player, such as building walls, big tree trunks, etc.
- group 1 means physics object that will block the player, but not the camera, such as small stones, thin posts, trees, etc.

void **SetSelectGroupIndex**(int *nGroupIndex*)

set the selection group index. if -1, it means that it was not selected. this function is equivalent to calling ParaSelection.AddObject(obj, nGroupID);

Parameters

- `nGroupIndex`: selection group index.

int **GetSelectGroupIndex**()

get the selection group index. if -1, it means that it was not selected.

void **SetDensity**(float *fDensity*)

body density. The water density is always 1.0 in the game engine. So if it is above 1.0, it will sink in water; otherwise it will float on water surface. So if it is below 0.5, it will fly or glind in air falling down with little gravity; otherwise it will fall down with full gravity. A density of 0 or negative, means that the character can fly. The default value is 1.2. the following are some examples

- character: body density: 1.2
- car: body density: 2.0 mount target
- ship: body density: 0.8 mount target
- plane: body density: 0.4 mount target

Parameters

- `fDensity`:

float **GetDensity**()

get body density

float **GetPhysicsRadius**()

the biped is modeled as a cylinder or sphere during rough physics calculation. this function returns the radius of the cylinder or sphere.

void **SetPhysicsRadius**(float *fR*)

the biped is modeled as a cylinder or sphere during rough physics calculation. this function set the radius of the cylinder or sphere.

float **GetPhysicsHeight**()

the biped is modeled as a cylinder or sphere during rough physics calculation. this function returns the height of the cylinder or sphere. this value will also restrict the biped's vertical movement. if there is no head attachment, the GetPhysicsHeight is used for character head on text position instead. If PhysicsHeight is never set, it is always 4*PhysicsRadius. However, if it is set, its value are maintained.

void **SetPhysicsHeight**(float *fHeight*)

the biped is modeled as a cylinder or sphere during rough physics calculation. this function set the height of the cylinder or sphere. this value will also restrict the biped's vertical movement. if there is no head attachment, the GetPhysicsHeight is used for character head on text position instead. If PhysicsHeight is never set, it is always 4*PhysicsRadius. However, if it is set, its value are maintained.

string **GetName**() **const**
get the object name

const char ***GetName_**() **const**
for .NET API use only.not thread safe.

void **SetName**(**const** char **sName*)
set the object name

Remark : currently, renaming an object after attaching it to the scene is dangerous, because when people use the *ParaScene.GetObject()*, it may return a renamed and deleted object. The good practice is that never change an object's name when it is attached to the scene.

Remark : In version 0.9 or above, renaming an object will detach the object from the scene and then reattach it to the scene using the new name; so it works fine when changing object name after attachment.

Remark : this function will not take effect, if the object can not be renamed, such as due to another global object with the same name.

Parameters

- `sName`: new name of the object

void **SnapToTerrainSurface**(int *bUseNorm*)
snap to terrain surface. Such as landscape trees, stones, etc. set bUseNorm to 1, if you want the norm to be aligned.

bool **IsCharacter**() **const**
Return return true if object is a character(biped) object

bool **IsOPC**() **const**
Return return true if object is a network player

ParaCharacter **ToCharacter**()
return the *ParaCharacter* object, if this object is a biped typed scene object. the *ParaCharacter* interface offers more specific functions to control the behavior and appearance of the character object.
See *ParaCharacter*

IGameObject ***ToGameObject**()
convert to game object. This function may return NULL.

void **AddEvent**(**const** char **strEvent*, int *nEventType*, bool *bIsUnique*)
add events to object to control the object, please refer to HLE for object event specifications. Generally, one can use event to tell a biped to walk to a new position play a certain animation, speak some words, follow another biped, attack, etc.

Parameters

- `strEvent`: currently a biped object accepts the following events. A list of events (more will be coming in future release):
  - "stop" Stop the biped
  - "walk x y" Walk to (x,y) using the default animation
  - "colr r, g, b" Set model color[0, 1] e.g. colr 1.0 0 0 = to set red color
  - "anim string | number" Play animation
  - "asai type parameters" Assign AI module of type type to the biped. Old modules are discarded Currently only support creature type AI module, with one parameter which can be ranged or melee unit. type: AIModuleCreatures=1, // respawning creatures that may be neural,or aggressive. They will not attack bipeds of any type, but may attack any team bipeds. AIModuleTeam=2, // Player Controllable biped AI module. it recognize creatures and members in other teams by their IDs. "asai 1 1" Assign creature AI with ranged unit attribute "asai 1 0" Assign creature AI with melee unit attribute
  - "ClearAll" Clear all tasks that this biped is assigned.

  - "task taskType parameters" Assign task to the AImodule associated with the biped. taskType: Currently taskType should be string, not their number
    * DieAndReborn = 0,
    * WanderNearby = 1,
    * Evade=2, "task Evade name" name: which biped to evade e.g. "task Evade enemy1"
    * Follow=3, "task Follow name" name: which biped to follow e.g. "task Follow LiXizhi"
    * Movie=4, "task Movie string" string: list of key, no spaces in the str are allowed. Format is given below [<const char* anim, float x, float y, float z, float facing, float duration>]+ If y=0, then the height of the position is ignored. Duration is in seconds. e.g. "task Movie <attack-1,50,0,60,0,10><Die-1,50,0,60,0,0>"
- `nEventType`: default to 0
- `bIsUnique`: is it unique by string

void **`AddChild`**(**const** *ParaObject* *obj*)
: attach the object as its child object.

  Parameters

  - `obj`: the child object to attach

void **`EnablePhysics`**(bool *bEnable*)
: if this is a physics mesh object, this function will turn on or off the physics of the object.

bool **`IsPhysicsEnabled`**()
: whether this is a physics mesh object

float **`DistanceTo`**(*ParaObject* *obj*)
: get the distance with another object

float **`DistanceToSq`**(*ParaObject* *obj*)
: get the distance square with another object

float **`DistanceToPlayerSq`**()
: get the distance square with the current player object

float **`DistanceToCameraSq`**()
: get the distance square with the camera object

object **`GetViewBox`**(**const** object *&output*)
: get the view box. e.g. local box = obj:GetViewBox({}); log(box.pos_x..box.pos_y..box.pos_z); return a table containing the following field:{pos_x, pos_y,pos_z,obb_x,obb_y,obb_z,} pos_x, pos_y,pos_z: is the point at the bottom center of the box. obb_x,obb_y,obb_z: is the size of the box.

bool **`HasAttachmentPoint`**(int *nAttachmentID*)
: return whether this model has a given attachment point

  Parameters

  - `nAttachmentID`: see ATTACHMENT_ID. default to 0, which is general mount point. ATT_ID_MOUNT1-9(20-28) is another mount points

void **`GetAttachmentPosition`**(int *nAttachmentID*, float **x*, float **y*, float **z*)
: return this object's attachment point e.g. local x,y,z = obj:GetAttachmentPosition(0);

  Return x,y,z: in world coordinates

  Parameters

- `nAttachmentID`: see ATTACHMENT_ID. default to 0, which is general mount point. ATT_ID_MOUNT1-9(20-28) is another mount points

void **`SetHomeZone`**(**const** char **sHomeZone*)
set the home zone of this object if any. it may return NULL, if zone is not visible.

Parameters

- `sHomeZone`: if this is NULL, it will remove the zone.

const char ***`GetHomeZone`**()
get the home zone of this object if any. it may return NULL or "", if zone is not visible.

void **`SetHeadOnText`**(**const** char **sText*, int *nIndex*)
set the text to be displayed on head on display

const char ***`GetHeadOnText`**(int *nIndex*)
Get the text to be displayed on head on display

void **`SetHeadOnUITemplateName`**(**const** char **sUIName*, int *nIndex*)
set which UI control the head on display will be used as a template for drawing the text it can be a single CGUIText Object or it can be a container with a direct children called "text" if this is "" or empty, the default UI template will be used. The default UI template is an invisible CGUIText control called "_HeadOnDisplayText_" By default, "_HeadOnDisplayText_" uses horizontal text alignment and system font.

const char ***`GetHeadOnUITemplateName`**(int *nIndex*)
get which UI control the head on display will be used as a template for drawing the text it can be a single CGUIText Object or it can be a container with a direct children called "text" if this is "" or empty, the default UI template will be used. The default UI template is an invisible CGUIText control called "_HeadOnDisplayText_" By default, "_HeadOnDisplayText_" uses horizontal text alignment and system font.

Return : it returns NULL if no UI head on display.

void **`SetHeadOnTextColor`**(**const** char **color*, int *nIndex*)
set the text to be displayed on head on display

Parameters

- `color`: "r g b" or "r g b a", such as "0 255 0", "0 255 0 255"

void **`SetHeadOnOffest`**(float *x*, float *y*, float *z*, int *nIndex*)
set the offset where head on display should be rendered relative to the origin or head of the host 3d object

void **`GetHeadOnOffset`**(int *nIndex*, float **x*, float **y*, float **z*)
Get the offset where head on display should be rendered relative to the origin or head of the host 3d object

void **`ShowHeadOnDisplay`**(bool *bShow*, int *nIndex*)
show or hide object's head on display

bool **`IsHeadOnDisplayShown`**(int *nIndex*)
whether the object head on display shall be visible

bool **`HasHeadOnDisplay`**(int *nIndex*)
whether the object contains head on display

int **`GetXRefScriptCount`**()
: get the number of the script X reference instances

const char ***`GetXRefScript`**(int *nIndex*)
: return xref script file path by index

void **`GetXRefScriptPosition`**(int *nIndex*, float **x*, float **y*, float **z*)
: get the 3D position in world space of the script object's origin

void **`GetXRefScriptScaling`**(int *nIndex*, float **x*, float **y*, float **z*)
: get the scaling of the object in both x,y,z directions

float **`GetXRefScriptFacing`**(int *nIndex*)
: get the facing of the object in xz plane

const char ***`GetXRefScriptLocalMatrix`**(int *nIndex*)
: get the local transform of the script object. It contains scale and rotation only the string returned by this function must be used at once and should not be saved. This function is also not thread-safe.

  Return : "11,12,13, 21,22,23, 31,32,33,41,42,43" ,where 41,42,43 are always 0. It may return NULL, if the local matrix does not exist.

bool **`IsSentient`**()
: whether the biped is sentient or not

float **`GetSentientRadius`**()
: get the sentient radius. usually this is much larger than the perceptive radius.

float **`GetPerceptiveRadius`**()
: get the perceptive radius.

void **`SetPerceptiveRadius`**(float *fNewRaduis*)
: Set the perceptive radius.

int **`GetNumOfPerceivedObject`**()
: return the total number of perceived objects.

ParaObject **`GetPerceivedObject`**(int *nIndex*)
: get the perceived object by index. This function may return NULL.

bool **`IsAlwaysSentient`**()
: whether the object is always sentient. The current player is always sentient

void **`SetAlwaysSentient`**(bool *bAlways*)
: set whether sentient.

void **`MakeSentient`**(bool *bSentient*)
: set the object to sentient.

  Parameters

  - `bSentient`: true to make sentient. if the object's sentient count is larger than 0, this function has no effect false, to remove the object from the sentient list.

void **`UpdateTileContainer`**()
: update the tile container according to the current position of the game object. This function is automatically called when a global object is attached.

void **`MakeGlobal`**(bool *bGlobal*)
: make the biped global if it is not and vice versa.

bool **IsGlobal**()
: whether the object is global or not.

void **SetGroupID**(int *nGroup*)
: set the group ID to which this object belongs to. In order to be detected by other game object. Object needs to be in group 0 to 31. default value is 0

void **SetSentientField**(DWORD *dwFieldOrGroup*, bool *bIsGroup*)
: set the sentient field. A bit field of sentient object. from lower bit to higher bits, it matches to the 0-31 groups.

See *SetGroupID()* if this is 0x0000, it will detect no objects. If this is 0xffff, it will detects all objects in any of the 32 groups. if this is 0x0001, it will only detect group 0.

Parameters

- `dwFieldOrGroup`: this is either treated as field or group,depending on the bIsGroup parameter.
- `bIsGroup`: if this is true, dwFieldOrGroup is treated as a group number of which will object will detect. if this is false, dwFieldOrGroup is treated as a bitwise field of which will object will detect.

bool **IsSentientWith**(**const** *ParaObject* *&pObj*)
: return true if the current object is sentient to the specified object. If the object is always sentient, this function will always return true.

void **SetMovableRegion**(float *center_x*, float *center_y*, float *center_z*, float *extent_x*, float *extent_y*, float *extent_z*)
: Set the region within which the object can move. This function is not fully implemented on a per object basis.

Note : currently it sets the global movable region of the character.

void **GetMovableRegion**(float **center_x*, float **center_y*, float **center_z*, float **extent_x*, float **extent_y*, float **extent_z*)
: get the region within which the object can move.. This function takes no parameters. input are not input, but pure output. In the script, we can call it as below cx,cy,cz,ex,ey,ez = biped:*GetMovableRegion()*; get the biped's position please note, y is the absolute position in world coordinate

See *SetMovableRegion()*

void **SetAnimation**(int *nAnimID*)
: Set the current animation id

Parameters

- `nAnimID`: 0 is default standing animation. 4 is walking, 5 is running. more information, please see AnimationID

int **GetAnimation**()
: get the scaling.

string **GetOnEnterSentientArea**() **const**
: when other game objects of a different type entered the sentient area of this object. This function will be automatically called by the environment simulator.

string **GetOnLeaveSentientArea**() **const**
: when no other game objects of different type is in the sentient area of this object. This function will be automatically called by the environment simulator.

string **GetOnClick**() **const**
: when the player clicked on this object. This function will be automatically called by the environment simulator.

void **On_Click**(DWORD *nMouseKey*, DWORD *dwParam1*, DWORD *dwParam2*)
: activate the OnClick

string **GetOnPerceived**() **const**
: when other game objects of a different type entered the perceptive area of this object. This function will be automatically called by the environment simulator.

string **GetOnFrameMove**() **const**
: called every frame move when this character is sentient. This is most likely used by active AI controllers, such as movie controller.

string **GetOnNetSend**() **const**
: during the execution of this object, it may send various network commands to the server or client. the network module will decide when to group these commands and send them over the network in one package. this function will be called when such network package is being prepared.

string **GetOnNetReceive**() **const**
: when the network module receives packages from the network and it is about a certain game object. Then this function will be automatically called. In this function, the game object may read the network packages and act accordingly.

int **GetEffectHandle**()
: Get the current shader handle used to render the object

See TechniqueHandle

void **SetEffectHandle**(int *nHandle*)
: Set the shader handle used to render the object. Please note, object will be immediately rendered using the newly assigned shader in the next frame. shade handle in the range [0,2048] is reserved for ParaEngine's internal usage. CAUTION: setting a shader whose input is incompatible with the object's internal data presentation will cause the application to close.

See TechniqueHandle

Parameters

- `nHandle`:

int **AddReference**(**const** *ParaObject* &*maker*, int *nTag*)
: add a new reference.

Return

Parameters

- `maker`:

int **DeleteReference**(**const** *ParaObject* &*ref*)
: delete a reference.

Return return REF_FAIL if reference not found. otherwise REF_SUCCEED

Parameters

- `ref`:

int **DeleteAllRefs**()
: Deletes all references of this object.

int **GetRefObjNum**()

get the total number of references

ParaObject **GetRefObject**(int *nIndex*)

get the referenced object at the given index

ParaScripting::*ParaAssetObject* **GetTexture**()

get primary texture object

int **GetNumReplaceableTextures**()

get the total number of replaceable textures, which is the largest replaceable texture ID. but it does not mean that all ID contains valid replaceable textures. This function can be used to quickly decide whether the model contains replaceable textures. Generally we allow 32 replaceable textures per model.

Note : This function will cause the mesh entity to be initialized.

Return 0 may be returned if no replaceable texture is used by the model.

ParaAssetObject **GetDefaultReplaceableTexture**(int *ReplaceableTextureID*)

get the default replaceable texture by its ID. The default replaceable texture is the main texture exported from the 3dsmax exporter.

Note : This function will cause the mesh entity to be initialized.

Return this may return NULL, if replaceable texture is not set before or ID is invalid.

Parameters

- `ReplaceableTextureID`: usually [0-32) generally speaking, replaceable ID 0 is used for general purpose replaceable texture, ID 1 is for user defined. ID 2 is for custom skins.

ParaAssetObject **GetReplaceableTexture**(int *ReplaceableTextureID*)

get the current replaceable texture by its ID. if no replaceable textures is set before, this will return the same result as *GetNumReplaceableTextures()*.

Note : This function will cause the mesh entity to be initialized.

Return this may return NULL, if replaceable texture is not set before or ID is invalid.

Parameters

- `ReplaceableTextureID`: usually [0-32) generally speaking, replaceable ID 0 is used for general purpose replaceable texture, ID 1 is for user defined. ID 2 is for custom skins.

bool **SetReplaceableTexture**(int *ReplaceableTextureID*, *ParaAssetObject* *pTextureEntity*)

set the replaceable texture at the given index with a new texture. this function will succeed regardless whether the mesh is initialized. Hence it can be used at loading time. because default instance of the mesh may use different replaceable texture set.

Return true if succeed. if ReplaceableTextureID exceed the total number of replaceable textures, this function will return false.

Parameters

- `ReplaceableTextureID`: usually [0-32) generally speaking, replaceable ID 0 is used for general purpose replaceable texture, ID 1 is for user defined. ID 2 is for custom skins.
- `pTextureEntity`: The reference account of the texture entity will be automatically increased by one.

class **ParaObjectNode**

#include <ParaScriptingGlobal.h> for Para Script global dictionary object

class **ParaPainter**

#include <ParaScriptingPainter.h> the current painter object. call Begin() End() to switch paint device. please note, for ownerdraw GUI callbacks, they are automatically called.

for performance reasons, all methods are static.

Public Static Functions

void **SetFont** (**const** object *&font*)

set current font

Parameters

- `font`: {family=”System”, size=10, bold=true} or it can be string “System;14;” or “System;14;bold”

void **SetPen** (**const** object *&pen*)

set current pen

Parameters

- `pen`: { width=1, brush = {color=”#00000000”, texture=”filename or texture asset”}, } or it can be {width=1, color=”#000000”, texture=”filename or texture asset”} or it can be pen color “#ff000000” or “255 255 255”

void **SetBrush** (**const** object *&brush*)

set current brush (texture and color)

Parameters

- `brush`: { color=”#00000000”, texture=”filename or texture asset”} or it can be pen color “#ff000000” or “255 255 255”

void **SetBackground** (**const** object *&brush*)

set current background brush

Parameters

- `brush`: { color=”#00000000”, texture=”filename or texture asset”}

void **SetOpacity** (float *fOpacity*)

between [0,1]

void **SetTransform** (**const** object *&trans*, bool *combine*)

Sets the world transformation matrix. If combine is true, the specified matrix is combined with the current matrix; otherwise it replaces the current matrix.

void **DrawTriangleList** (**const** object *&triangleList*, int *nTriangleCount*, int *nIndexOffset*)

draw 3d triangles

Parameters

- `triangleList`: array of triangle vertices {{x,y,z}, {x,y,z}, {x,y,z}, ...},
- `nTriangleCount`: triangle count.
- `nIndexOffset`: start index offset. default to 0.

void **DrawLineList** (**const** object *&lineList*, int *nLineCount*, int *nIndexOffset*)

draw 3d lines

Parameters

- `lineList`: array of line vertices {{x,y,z}, {x,y,z}, ...},
- `nTriangleCount`: triangle count
- `nIndexOffset`: start index offset. default to 0.

class **`ParaParamBlock`**

#include <ParaScriptingCommon.h> a list of CParameter{name, value} pairs of anything. usually used for DirectX effect parameter block. value can be integer, float, vector3, vector4, matrix, TextureEntity, etc.

Public Functions

bool **`IsValid`**()

check if the object is valid

void **`Clear`**()

clear all parameters

void **`SetMatrix43`**(**const** char **sParamName*, **const** char **matrix*)

set matrix by a string of 4*3 number of float values separated by comma (see below): "mat._11, mat._12, mat._13, mat._21, mat._22, mat._23,mat._31, mat._32, mat._33,mat._41, mat._42, mat._43" If a blank string("") is specified, identity matrix is set

void **`SetParam`**(**const** char **sParamName*, **const** char **sValue*)

setting known parameters to its predefined or current value.

Parameters

- `sValue`: known values such as "mat4Project", "mat4ProjectionInverse", "mat4Projection", "mat4ModelView", "mat4ModelViewInverse", "mat4ShadowMapTex", "vec3cameraPosition"

void **`SetTexture`**(int *nTextureIndex*, **const** char **sFilePath*)

set texture with the given texture index,

Parameters

- `nTextureIndex`: usually [0,9], which texture register to use in case of effect file parameter.
- `sFilePath`: the file name of the texture.

void **`SetTextureObj`**(int *nTextureIndex*, **const** *ParaAssetObject* &*assetObject*)

same as SetTexture, except that *ParaAssetObject* is an object.

class **`ParaScene`**

#include <ParaScriptingScene.h> *ParaScene* namespace contains a list of HAPI functions to create and modify scene objects in paraworld. The following are basic steps to create scene object:

- Use Create*() functions to create scene object. The created object is return as a *ParaObject* instance.
- Use the *ParaObject* instance to modify the position and orientation of the object.
- Use Attach method to insert the scene object to the scene.

Please note: if an object is created without attaching to the scene, it may result in memory leak. although, we may automatically clear unattached scene object, when the application quit.

Public Static Functions

ParaAttributeObject **GetAttributeObject**()
get the attribute object associated with the global scene.

void **GetAttributeObject_**(*ParaAttributeObject &output*)
used for API exportation.

ParaAttributeObject **GetAttributeObjectSky**()
get the attribute object associated with the current sky object.

void **GetAttributeObjectSky_**(*ParaAttributeObject &output*)
used for API exportation.

ParaAttributeObject **GetAttributeObjectPlayer**()
get the attribute object associated with the current player.

void **GetAttributeObjectPlayer_**(*ParaAttributeObject &output*)
used for API exportation.

ParaAttributeObject **GetAttributeObjectOcean**()
get the attribute object associated with the global ocean manager.

void **GetAttributeObjectOcean_**(*ParaAttributeObject &output*)
used for API exportation.

ParaAttributeObject **GetAttributeObjectSunLight**()
get the attribute object associated with the sun light .

void **GetAttributeObjectSunLight_**(*ParaAttributeObject &output*)
used for API exportation.

ParaObject **GetObject**(**const** char **strObjName*)
get the scene object by name. currently, the object must be global, in order to be found by its name.

Remark : if local mesh's name begins with "g_", it can also be retrieved by calling this function. however, if a global object has the same name, the global object is always returned instead of the local mesh.

Parameters

- `strObjName`: the format of the name is as below: strObjName := [<_type>]string _type := managed_loader | OPC | NPC | player | zone | portal e.g. strObjName = "creatures1" or "<managed_loader>sceneloader1" or "<player>".

ParaScripting::*ParaObject* **GetObject5**(int *nID*)
get an object by its ID

ParaObject **GetObject3**(float *x*, float *y*, float *z*)
get the first local object,whose position is very close to vPos. This function will search for the first (local mesh) object throughout the hierachy of the scene. this function is kind of slow, please do not call on a per frame basis. Use *GetObjectByViewBox()* to get an object faster.

Return : NULL if not found

Parameters

- `vPos`: world position of the local mesh object
- `fEpsilon`: if a mesh is close enough to vPos within this value.

ParaObject **`GetObject4`** (float *x*, float *y*, float *z*, float *fEpsilon*)
get the first object that matches the position within fEpsilon, which is usually 0.01f

void **`GetObject_`** (*ParaObject* **pOut*, **const** char **strObjName*)
this function shall never be called from the scripting interface. this is solely for exporting API. and should not be used from the scripting interface.

ParaObject **`GetPlayer`** ()
Get the current player. same as *ParaScene.GetObject*("<player>").

void **`GetPlayer_`** (*ParaObject* **pOut*)
this function shall never be called from the scripting interface. this is solely for exporting API. and should not be used from the scripting interface.

ParaObject **`GetNextObject`** (*ParaObject* &*obj*)
get the next scene object.

Return : return the next object. the returned object is invalid if there is only one object left.

Parameters

- `obj`: the object whose next object is retrieved.

void **`CreateWorld`** (**const** char **sWorldName*, float *fWorldSize*, **const** char **sConfigFile*)
Create a new parallel world of a given size. When this function is called, it will replace previously created world of the same name. Currently only a single world can be created at any given time. In future, we will support hosting several world simultaneously.

Parameters

- `sWorldName`: name of the world to be created.
- `fWorldSize`: the size of the world in meters.
- `sConfigFile`: the file name of the configuration file. Currently it is the same as the terrain configuration file

void **`Reset`** ()
reset the scene to blank. this function is NOT automatically called when a new isolated world is created. so one need to call *Reset()* when it wants to change the world, otherwise the new world will be merged into the previous world.

See *CreateWorld()*

ParaObject **`CreateManagedLoader`** (**const** char **sLoaderName*)
Create a managed loader for dynamic scene object loading and unloading. The behavior of a managed loader is below:

- The child objects of a managed loader will be automatically loaded and unloaded as a single entity.
- Generally only static objects are attached to a managed loader.
- Different managed loaders in a *ParaScene* must have different names.
- if one create a manager loader with the same name several times, the same managed loader will be returned.
- the bounding box of a managed loader will be automatically calculated as new child objects are attached to it.

- The current loader algorithm will linearly transverse all managed loaders in the scene to decide which scene objects to load or unload. Although the routine is executed when the CPU is free, it is good practice to keep the total number of managed loaders in a single scene low. I think a couple of thousand loaders will be fine for current hardware.
- it is good practice to use managed loaders to group all static scene objects that appears in a scene. Because, it will keep the scene graph to a moderate size automatically and accelerate physics calculation, etc.

The following NPL code shows typical usage of the managed loader. Generally a managed loader and its children are written in a single script file. Then, any other script can call dofile() or NPL.ActivateCopy() to run the script as many times as they like. The code however will ensure that objects managed by the loader will only be created and attached once in the game engine.There may be a setting in ParaEngine to do automatic garbage collection with managed loaders, so one may need to call the following script file often enough for the managed objects to stay active in the game scene. local sceneLoader = *ParaScene.GetObject*("<managed_loader>scene2"); if (sceneLoader:IsValid() == true) then if the scene loader already exists, just attach it to the scene. ParaScene.Attach(sceneLoader); else if the scene loader is not created before, we will create it now sceneLoader = *ParaScene.CreateManagedLoader*("scene2"); ParaAsset.ParaXModel("tiny", "Units/Human/Peasant/peasant.x"); create scene objects and add them to managed loader object local player; player = *ParaScene.CreateCharacter* ("LiXizhi2", "tiny", "", false, 0.5, 0, 1.0); player:SetPosition(21758, 0, 16221); player:SnapToTerrainSurface(0); sceneLoader:AddChild(player); attach all objects in the loader to the scene graph ParaScene.Attach(sceneLoader); end

See CManagedLoaderObject in ParaEngine reference.

Parameters

- `sLoaderName`: the name of the loader.

void **`CreateGlobalTerrain`** (float *fRadius*, int *nDepth*, **const** char **sHeightmapfile*, float *fTerrainSize*, float *fElevscale*, int *bSwapvertical*, **const** char **sMainTextureFile*, **const** char **sCommonTextureFile*, int *nMaxBlockSize*, float *fDetailThreshold*)

update the terrain tile at a given tile location. In case a world is extremely large, it can be divided into a matrix of square tiles of a given size. there is no limitation on the size of this matrix, hence the (nRow, nCol) parameter can be any integer pair. but typically, the matrix is rarely larger than 64*64. We use indexed set to save the matrix, so the memory consumption of the matrix is linearly proportionally to the number of terrain tiles created.

See CreateWorld(const char * sWorldName, float fWorldSize, float fTerrainTileSize); Create and set the global terrain from height map and texture files. this function can be called multiple times, in which cases previously loaded terrain will be discarded example: *ParaScene.CreateGlobalTerrain*(2048, 7, "LlanoElev.png", 5.0, 15.0, 1, "LlanoTex.jpg", "dirt2.jpg", 64, 10.0");

Parameters

- `nRow`: the row number in the terrain tile matrix.nRow>=0.
- `nCol`: the row number in the terrain tile matrix.nCol>=0.
- `sConfigFile`: the terrain tile configuration file to be used to create the terrain at (nRow, nCol). if sConfigFile == "", then the terrain at (nRow, nCol) will be removed.

Parameters

- `fRadius`: entire terrain size, this has doing to do with the actual terrain map size, it just prevent mobile characters from walking outside it.

- `nDepth`: depth of the quad tree terrain hierarchy. objects created on the terrain will be organized in a quad tree. This is the depth of the quad tree. It should not be too big. usually 7 is enough. the rest of the parameters specify the data to render the terrain.
- `sHeightmapfile`: the height map used to create the terrain. It must be sized to 2*2*...*2 pixels for both height and width. so usually it is 1024*1024, 2048*2048, etc.
- `fTerrainSize`: the actual terrain size in the game
- `bSwapvertical`: if one want to swap the height map data vertically.
- `sMainTextureFile`: texture to be mapped to entire terrain
- `sCommonTextureFile`: texture to be tiles to the entire terrain to add some details.
- `nMaxBlockSize`: When doing LOD with the height map, the max block size must be smaller than this one. This will be (nMaxBlockSize*nMaxBlockSize) sized region on the height map.
- `fDetailThreshold`: we will use a LOD block to approximate the terrain at its location, if the block is smaller than fDetailThreshold pixels when projected to the 2D screen.

void **`Attach`**(*ParaObject* &*pObj*)

Automatically attach a scene object to the scene graph according to its type and position. The object can be a manager loader, a global object or any ordinary scene object.

•For tiled object, it is added to the smallest CTerrainTile in the quad-tree

•For global tiled object, it is added to the root CTerrainTile

•For non-tiled object, it is added to an automatically created CContainerObject whose name is the class identifier name of the object. hence objects are automatically grouped by class type on the root scene's child nodes. To explicitly add an object to a specified parent, use AddChild() method on the parent node.

Return : parent object is returned if successfully attached. For tiled object, this is the smallest terrain tile that contains the object. For non-tiled object, this is the automatically created CContainerObject that.

Note : If the object has already been attached to the scene, it will be removed and reattached. In most cases, a strong reference of the object is kept by its parent.

void **`Delete`**(*ParaObject* &*pObj*)

delete the object. If the object is root scene object, then the entire scene is deleted.

void **`Detach`**(*ParaObject* &*pObj*)

detach the object. Be sure that the object is properly deleted after it is detached from the scene, because the scene graph is not responsible to manage it any more. The only exception is the managed loader object.

See *CreateManagedLoader(const char * sLoaderName)*

void **`FireMissile`**(int *nMissileID*, float *fSpeed*, double *fromX*, double *fromY*, double *fromZ*, double *toX*, double *toY*, double *toZ*)

fire a missile from (fromX, fromY, fromZ) to (toX, toY, toZ) using the specified missile object and speed.

void **`SetModified`**(bool *bModified*)

set whether scene is modified

bool **`IsModified`**()

Get whether scene is modified

bool **IsScenePaused**()
: when a scene is paused, all animation will be frozen.

void **PauseScene**(bool *bEnable*)
: pause the scene

bool **IsSceneEnabled**()
: whether 3D scene is enabled or not. a disabled scene is not visible no matter what. This function must be called at least once whenever a new scene is loaded, or 3D scene will not be displayed. A scene is automatically disabled when cleaned up.

void **EnableScene**(bool *bEnable*)
: enable the scene

ParaObject **CreateMeshObject**(**const** char **strObjectName*, **const** char **strMeshAssetName*, float *fOBB_X*, float *fOBB_Y*, float *fOBB_Z*, float *fFacing*, bool *bSolid*, **const** char **localMatrix*)
: create a simple and static mesh object in the scene. Solid mesh will collide with mobile characters. Simple mesh does not implement clipping optimization, hence should not be very large. It is good to use it

Parameters

- `strMeshAssetName`: for small houses, trees, stones, small hills, etc. when naming mesh file, one can combine "_a"(no physics), "_b"(billboard), "_t"(transparent), "_d"(dim or no lighting), "_r"(receive shadow) in the file name in any order, such as "xxx_b_t_d.x". all such special file endings are listed below
  - "_a": no physics, it will have no physics, even bApplyPhysics is true. For example. "grass_a.x".
  - "_b": billboarded and no physics
  - "_r": mesh is shadow receiver
  - "_e": mesh is not a shadow caster
  - "_t": mesh contains majority transparent objects. Please note that this is different from alpha testing. Transparency throguh alpha testing is not transparent.
  - "_p": mesh is picture or particles. They are rendered with billboarding and Force No Light flag on.
  - "_d": mesh is dim and is rendered with force no light on.
  - "_v": mesh is rendered with vegetation shader. the static model like palm trees, grasses, bamboos can be animated by this shader.
- `fOBB_X`: object bounding box.x
- `fOBB_Y`: object bounding box.y
- `fOBB_Z`: object bounding box.z
- `fFacing`: rotation of the bounding box around the y axis.
- `bSolid`: 1, if it is a solid mesh, otherwise it is passable.
- `localMatrix`: the local transformation matrix of the mesh. It is a string of 4*3 number of float values separated by comma (see below): "mat._11, mat._12, mat._13, mat._21, mat._22, mat._23,mat._31, mat._32, mat._33,mat._41, mat._42, mat._43" If a blank string("") is specified, the local matrix is set to identity matrix.

ParaScripting::*ParaObject* **`CreateObject`** (**const** char **strType*, **const** char **strObjectName*, double *x*, double *y*, double *z*)

create an object according to type.

Return *ParaObject*

Parameters

- `strType`: as returned by GetAttributeClassName of IAttributeField, such as "CMeshPhysicsObject", "CMeshObject", "CBipedObject", etc.
- `strObjectName`: string identifier of the object
- `xyz`: position of the object.

ParaScripting::*ParaObject* **`CreateZone`** (**const** char **sZoneName*, **const** char **sBoundingVolumes*, float *width*, float *height*, float *depth*, float *facing*)

create a portal zone object for portal rendering.

Parameters

- `sZoneName`: it must be unique.
- `fRadius`: the zone sphere radius is an approximation of the bounding volume. we will only further check if an object is inside a zone, if it is first inside this sphere.
- `sBoundingVolumes`: if this is "", the zone will not be able to automatically determine which mobile objects are in it. or it can be "x1,y1,z1;x2,y2,z2;x3,y3,z3;" each three value is a point in local space denoting a plane of the bounding volume. because the convex bounding volume in local space always contains the origin, three values is enough to represent a plane in the bounding volume.
- `widthheightdepthfacing`: the bounding shape of the portal.

ParaScripting::*ParaObject* **`CreatePortal`** (**const** char **sPortalName*, **const** char **sHomeZone*, **const** char **sTargetZone*, **const** char **sQuadVertices*, float *width*, float *height*, float *depth*, float *facing*)

create a portal object for portal rendering

Parameters

- `sPortalName`: it must be a unique name.
- `sHomeZone`: a zone name that this portal connects. this can be "", if a portal is not connected to any zone.
- `sTargetZone`: another zone name that this portal connects. this can be "", if the portal is connected to outer space.
- `sQuadVertices`: it is coordinates of the 4 quad vertices, "x1,y1,z1;x2,y2,z2;x3,y3,z3;x4,y4,z4;" the order of the first three vertices decides the direction of the quad plane. direction of quad plane is only useful when the portal's sTargetZone is empty(outer space), and it should always point from home zone to outer space.
- `widthheightdepthfacing`: the bounding size of the portal.

ParaScripting::*ParaObject* **`CreateVoxelMesh`** (**const** char **strObjectName*, **const** char **sGridFileName*, **const** char **sTextureFileName*)

create a voxel mesh object. A voxel mesh is a 32*32*32 grid which can be editable at runtime, and rendered using the matching cube algorithm.

Parameters

- `strObjectName`:
- `sGridFileName`: the file name from which to load.This can be "", where an empty one is created.
- `sTextureFileName`: the texture to use, one can later change this by calling SetReplaceableTexture.

bool **`CreateSkyBox`** (**const** char **strObjectName*, **const** char **strMeshAssetName*, float *fScaleX*, float *fScaleY*, float *fScaleZ*, float *fHeightOffset*)

create a sky box and add it to the current list. sky box with the same name will not be recreated,but will be selected as the current sky box. It may be a sky box/dome/plane or whatever. The associated mesh will be scaled by the specified amount along x,y,z axis and then translate up or down along the y axis. in many cases, the mesh data in the mesh asset is of unit size.

Parameters

- `strObjectName`: sky name
- `strMeshAssetName`: mesh asset name. this is not the file name.
- `fScaleX`: the static mesh local transform scale along the x axis
- `fScaleY`: the static mesh local transform scale along the y axis
- `fScaleZ`: the static mesh local transform scale along the z axis
- `fHeightOffset`: the translation along the y axis.

void **`DeleteSkyBox`** (**const** char **strObjectName*)

delete a name sky box.

Parameters

- `strObjectName`: if this is "", all sky boxes will be deleted.

ParaObject **`CreateMeshPhysicsObject`** (**const** char **strObjectName*, **const** char **strMeshAssetName*, float *fOBB_X*, float *fOBB_Y*, float *fOBB_Z*, bool *bApplyPhysics*, **const** char **localMatrix*)

Create static triangle mesh based actor for novodex physics engine. Some notes about meshes:

- Be sure that you define face normals as facing in the direction you intend. Collision detection will only work correctly between shapes approaching the mesh from the outside, i.e. from the direction in which the face normals point.
- Do not duplicate identical vertices! If you have two triangles sharing a vertex, this vertex should only occur once in the vertex list, and both triangles should index it in the index list. If you create two copies of the vertex, the collision detection code wont know that it is actually the same vertex, which leads to a decreased performance and unreliable results.
- Also avoid t-joints and non-manifold edges for the same reason. (A t-joint is a vertex of one triangle that is placed right on top of an edge of another triangle, but this second triangle is not split into two triangles at the vertex, as it should. A non-manifold edge is an edge (a pair of vertices) that is referenced by more than two triangles.)

Parameters

– `strMeshAssetName`: the mesh asset name which contains the triangular mesh. when naming mesh file, one can combine "_a"(no physics), "_b"(billboard), "_t"(transparent), "_d"(dim or no lighting), "_r"(receive shadow) in the file name in any order, such as "xxx_b_t_d.x". all such special file endings are listed below

"_a": no physics, it will have no physics, even bApplyPhysics is true. For example. "grass_a.x".

- "_b": billboarded and no physics
- "_r": mesh is shadow receiver
- "_e": mesh is not a shadow caster
- "_t": mesh contains majorily transparent objects. Please note that this is diferent from alpha testing. Transparency throguh alpha testing is not transparent.
- "_p": mesh is picture or particles. They are rendered with billboarding and Force No Light flag on.
- "_d": mesh is dim and is rendered with force no light on.
- "_v": mesh is rendered with vegetation shader. the static model like palm trees, grasses, bamboos can be animated by this shader.

Parameters

- `fOBB_X`: object bounding box.x
- `fOBB_Y`: object bounding box.y
- `fOBB_Z`: object bounding box.z
- `bApplyPhysics`: whether to turn on the physics of the mesh
- `localMatrix`: the local transformation matrix of the mesh. It is a string of 4*3 number of float values separated by comma (see below): "mat._11, mat._12, mat._13, mat._21, mat._22, mat._23,mat._31, mat._32, mat._33,mat._41, mat._42, mat._43" If a blank string("") is specified, the local matrix is set to identity matrix.

void **`CreateMeshPhysicsObject__`**(*ParaObject* **pOut*, **const** char **strObjectName*, *ParaAssetObject* *&asset*, float *fOBB_X*, float *fOBB_Y*, float *fOBB_Z*, bool *bApplyPhysics*, **const** char **localMatrix*)

this function shall never be called from the scripting interface. this is solely for exporting API. and should not be used from the scripting interface.

ParaObject **`CreateLightObject`**(**const** char **strObjectName*, float *fPosX*, float *fPosY*, float *fPosZ*, **const** char **sLightParams*, **const** char **localMatrix*)

Create a new light object. This is used to make light objects persistent.

Return light object is created and returned. One need to attach it to scene.

Parameters

- `strObjectName`: Light Object's name
- `fPosX`: world position X
- `fPosY`: world position Y
- `fPosZ`: world position Z
- `sLightParams`: if this is "" a default light will be created. otherwise it is in the following format. format is "Type Range (r g b a) att0 att1 att2" D3DLIGHTTYPE Type; Type of light source
  - D3DLIGHT_POINT = 1,
  - D3DLIGHT_SPOT = 2,
  - D3DLIGHT_DIRECTIONAL = 3, float Range; Cutoff range D3DCOLORVALUE Diffuse; Diffuse color of light float Attenuation0; Constant attenuation float Attenuation1; Linear attenuation float Attenuation2; Quadratic attenuation e.g. "1 7.0 (1 1 0 1) 0.3 0.1

1" light intensity is calculated as 1/(Attenuation0+d*Attenuation1+d*d*Attenuation2), where d is the distance from the light to object center.

Parameters

- `localMatrix`: the local transformation matrix of the mesh. It is a string of 4*3 number of float values separated by comma (see below): "mat._11, mat._12, mat._13, mat._21, mat._22, mat._23,mat._31, mat._32, mat._33,mat._41, mat._42, mat._43" If a blank string("") is specified, the local matrix is set to identity matrix.

ParaObject **`CreateDynamicPhysicsObject`**(**const** char **strObjectName*, **const** char **strMeshAssetName*, float *fOBB_X*, float *fOBB_Y*, float *fOBB_Z*, bool *bRenderMesh*)

Create dynamic physics actor for novodex physics engine. dynamic objects are considered global object and is loaded to the physics engine immediately currently, only Box and sphere shaped objects are supported.

ParaObject **`CreateCharacter`**(**const** char **strObjectName*, **const** char **strMultiAnimationAssetName*, **const** char **strScript*, bool *bIsGlobal*, float *fRadius*, float *fFacing*, float *fScaling*)

Create Character.

Parameters

- `strObjectName`: the name short cut for this object. If the character with the same name exists, it will be renamed to a random name. So that the character is always created.
- `strMultiAnimationAssetName`: the asset name of the base model. It can be nil, in which one must specified it in other means. if the asset file name ends with "_s", it will always be static and local , even IsGlobal is true. For example. "windmill_s.x". Some other special file endings are listed below
  - "_s": force static, solid and local.
- `strScript`: The script file to be loaded when the object is loaded for the first time
- `bIsGlobal`: a global character is not attached to the quad tree terrain, thus can move around the entire scene a non-global character is attached to the quad tree terrain, and is generally considered immobile. although it is OK for non-global character to move only in very small region.
- `fRadius`: radius of the character used in collision detection. if radius is 0, then the object is regarded as passable (not solid).
- `fFacing`: rotation around the y axis
- `fScaling`: typically this should be 1, however, one can scale the mesh and animation to render the character in a different size.

void **`CreateCharacter__`**(*ParaObject* **pOut*, **const** char **strObjectName*, *ParaAssetObject* *&asset*, **const** char **strScript*, bool *bIsGlobal*, float *fRadius*, float *fFacing*, float *fScaling*)

this function shall never be called from the scripting interface. this is solely for exporting API. and should not be used from the scripting interface.

void **`Play3DSound`**(**const** char **strSoundAssetName*, float *fX*, float *fY*, float *fZ*)

play a 3D sound at world location (fx, fy, fz). Currently, the sound is played until it stopped(no looping). And the sound range is internally defined. Please use the *ParaUI.PlaySound()* to play an ordinary sound or music with or without looping.

void **`SetGlobalWater`**(bool *bEnable*, float *fWaterLevel*)
Set the global water drawing attribute. the global water level is just a water surface at a given height, which is always drawn at current camera location to fill the entire screen. Water surface will be drawn after terrain

Parameters

- `bEnable`: whether to draw global water
- `fWaterLevel`: water level in meters. Default value is 0.0f;

float **`GetGlobalWaterLevel`**()
get the current global water level in meters. This function will return valid value even if the water is disabled.

bool **`IsGlobalWaterEnabled`**()
return true if global ocean water is enabled.

void **`UpdateOcean`**()
call this function, when the ocean has changed or the environment has changed. This will cause the reflection map of the ocean surface to redraw regardless of whether the camera moves or not.

void **`AddWaterRipple`**(float *x*, float *y*, float *z*)
add water ripple to the ocean surface.

See *GetGlobalWaterLevel()*

Parameters

- `x`: position of the ripple
- `y`: this is usually set to the current water level.

Parameters

- `z`: position of the ripple

void **`Execute`**(**const** char **strCmd*)
execute the scene command string. Most commands are for debugging purposes. The list of command is given below:

- "show OBB" display every scene object's bounding box
- "hide OBB" hide every scene object's bounding box
- "show report" display the per frame report, such as number of mesh drawn, number of terrain triangles, etc,..
- "hide report" hide the per frame report, such as number of mesh drawn, number of terrain triangles, etc,.. strCmd: the string command passed

ParaObject **`MousePick`**(float *fMaxDistance*, **const** char **sFilterFunc*)
Pick scene object at the current mouse cursor position. pick the smallest intersected object which is un-occluded by any objects. Object A is considered occluded by object B only if (1) both A and B intersect with the hit ray. (2) both A and B do not intersect with each other. (3) B is in front of A, with regard to the ray origin.

this function will ray-pick any loaded scene object(biped & mesh, but excluding the terrain) using their oriented bounding box. a filter function may be provided to further filter selected object. this function will transform all objects to near-camera coordinate system. This will remove some floating point inaccuracy near the camera position.Hence this function is most suitable for testing object near the camera eye position. This function does not rely on the physics engine to perform ray-picking.

fMaxDistance: the longest distance from the ray origin to check for collision. If the value is 0 or negative, the view culling radius is used as the fMaxDistance.

Return :the scene object. if the object is invalid, it means that the ray has hit nothing.

Parameters

- `sFnctFilter`: it can be any of the following string or a number string "mesh": mesh any mesh object in the scene. Usually for selection during scene editing. "cmesh": mesh object that is clickable (associated with scripts). Usually for game playing. "notplayer": any object in the scene except for the current player. Usually for selection during scene editing. "": any object in the scene except. Usually for selection during scene editing. "light": only pick light objects "biped": any character objects :local or global. "anyobject": any objects, including mesh and characters. but not including helper objects, such as light. "global": all global objects, such as global character and mesh. This is usually for game mode. "point": the returned object is invalid if there no collision with any physics faces. otherwise, one can use GetPosition function of the returned object to retrieve the intersection point. "actionmesh": mesh with action script. number: if it is a number, it is treated as a 32 bitwise DWORD filter code. see SetPickingFilter() for more example.

int **SelectObject** (int *nGroupIndex*, float *x*, float *y*, float *z*, float *radius*, **const** char **sFilterFunc*)
select select objects within a given region into a given group.

Return : the total number of selected objects is returned.

Parameters

- `nGroupIndex`: which group to select to. One can get the result from *ParaSelection*. In most cases, select to group 1; since group 0 is reserved for current selection.
- `xyzradius`: a sphere in world space.
- `sFnctFilter`: it can be any of the following string. "mesh": mesh any mesh object in the scene. Usually for selection during scene editing. "cmesh": mesh object that is clickable (associated with scripts). Usually for game playing. "notplayer": any object in the scene except for the current player. Usually for selection during scene editing. "": any object in the scene except. Usually for selection during scene editing. "light": only pick light objects "biped": any character objects :local or global. "anyobject": any objects, including mesh and characters. but not including helper objects, such as light. "global": all global objects, such as global character and mesh. This is usually for game mode.

int **SelectObject1** (int *nGroupIndex*, float *x1*, float *y1*, float *z1*, float *x2*, float *y2*, float *z2*, float *fRotY*, **const** char **sFilterFunc*)

Parameters

- `x1y1z1x2y2z2fRotY`: a bounding box: two diagonal points in world space and rotation around Y.

void **RegisterEvent** (**const** char **sID*, **const** char **sScript*)
register a mouse or key event handler

Parameters

- `sID`: a string identifier of the event handler. if sID begins with "_m" it is treated as a mouse click event, except that if sID begins with "_mm" it is treated as a mouse move event. if sID begins with "_md" it is treated as a mouse down event. if sID begins with "_mu" it is treated as a mouse up event. if sID begins with "_k" it is treated as a key down event. if sID begins with "_ku" it is treated as a key up event. if sID begins with "_n" it is treated as a network event handler. Note: mouse click is rarely used, since it can be produced in NPL via mouse down move and up. However, lazy NPL programmer can still use it if they do not like to write other mouse handlers in NPL.

- `sScript`: the script to be executed when the event is triggered.This is usually a function call in NPL. sScript should be in the following format "{NPL filename};{sCode};". this is the same format in the UI event handler

void **`RegisterEvent1`** (DWORD *nEventType*, **const** char **sID*, **const** char **sScript*)
same as above *RegisterEvent()*, except that it allows caller to explicitly specify the event type, instead of deriving it from the event name.

Parameters

- `nEventType`: any bit combination of EventHandler_type
- `sID`: any unique string identifier
- `sScript`: the NPL script.

void **`UnregisterEvent`** (**const** char **sID*)
unregister a mouse or key event handler

void **`UnregisterAllEvent`** ()
unregister all mouse or key event handler

void **`EnableMouseClick`** (bool *bEnable*)
whether the game engine will automatically generate mouse events for Game Objects.If true, The OnClick callback will be automatically called whenever there is a mouse click.

ParaObject **`GetCurrentActor`** ()
get the actor that is being processed by the AI module or a script call back. The validity of the pointer is not guaranteed.

void **`SetCurrentActor`** (*ParaObject* *pActor*)
Set the actor that is being processed by the AI module or a script call back. The pointer can be NULL.

ParaObject **`TogglePlayer`** ()
It changes the current player to the next player in the scene. this function is mostly for testing and game purpose. if the object has a reference object, the reference object will become the current object. return : the new current player is returned.

ParaObject **`GetObjectByViewBox`** (**const** object &*viewbox*)
get an object(usually a static mesh object) by a given view box.

Return : return the object with the closest match with the bounding box.

Parameters

- `viewbox`: One can get the view box by calling *ParaObject*:GetViewBox(). Or one can construct it using a table with the following field:{pos_x, pos_y,pos_z,obb_x,obb_y,obb_z,} pos_x, pos_y,pos_z: is the point at the bottom center of the box. obb_x,obb_y,obb_z: is the size of the box.

int **`GetActionMeshesBySphere`** (**const** object &*inout*, float *x*, float *y*, float *z*, float *radius*)
get the action meshes within or intersect a sphere. same as GetObjectsBySphere(..., "actionmesh")

int **`GetObjectsBySphere`** (**const** object &*inout*, float *x*, float *y*, float *z*, float *radius*, **const** char **sFilterFunc*)

Parameters

- `inout`: input and output, it should be an empty table.
- `x`: sphere center x
- `y`: sphere center y
- `z`: sphere center z
- `radius`: sphere radius
- `sFnctFilter`: it can be any of the following string. "mesh": mesh any mesh object in the scene. Usually for selection during scene editing. "cmesh": mesh object that is clickable (associated with scripts). Usually for game playing. "notplayer": any object in the scene except for the current player. Usually for selection during scene editing. "": any object in the scene except. Usually for selection during scene editing. "light": only pick light objects "biped": any character objects :local or global. "anyobject": any objects, including mesh and characters. but not including helper objects, such as light. "actionmesh": mesh with action script. "global": all global objects, such as global character and mesh. This is usually for game mode.

int **`GetObjectsByScreenRect`** (**const** object &*inout*, int *left*, int *top*, int *right*, int *bottom*, **const** char **sFilterFunc*, float *fMaxDistance*)

Get objects inside or intersect with a screen rect. screen rect is translated to a 3d cone from the camera eye position to a plane fMaxDistance away. This function is usually used for finding other static mesh objects near a certain character. please note that: objects must be completely inside the near and far planes in order to pass the test.

Return : return the number of objects in sphere.

Parameters

- `output`: list to get the result
- `lefttoprightbottom`: the rect in screen space coordinates
- `sFnctFilter`: it can be any of the following string. "mesh": mesh any mesh object in the scene. Usually for selection during scene editing. "cmesh": mesh object that is clickable (associated with scripts). Usually for game playing. "notplayer": any object in the scene except for the current player. Usually for selection during scene editing. "": any object in the scene except. Usually for selection during scene editing. "light": only pick light objects "biped": any character objects :local or global. "anyobject": any objects, including mesh and characters. but not including helper objects, such as light. "actionmesh": mesh with action script. "global": all global objects, such as global character and mesh. This is usually for game mode.
- `fMaxDistance`: the world unit distance inside which we will test possible candidates. if negative, it will get all visible ones.

void **`OnTerrainChanged`** (float *x*, float *y*, float *fRadius*)

This function is called manually to update characters in a square region. So that when the terrain heightfield is changed, the characters within the region will act accordingly, either falling down or snap to terrain surface.

Parameters

- `x`: center of the terrain region being modified.
- `y`: center of the terrain region being modified.
- `fRadius`: : radius of the terrain region being modified.

int **`SaveAllCharacters`**()
save all local characters in the scene to the current NPC database regardless of whether they are modified or not this function is usually called manually in some very rare cases. In most cases, call SaveLocalCharacters() instead.

Return the number of saved characters are returned.

void **`ShowHeadOnDisplay`**(bool *bShow*)
show or hide all scene's objects' head on display

bool **`IsHeadOnDisplayShown`**()
whether all scene's objects' head on display

int **`LoadNPCsByRegion`**(float *min_x*, float *min_y*, float *min_z*, float *max_x*, float *max_y*, float *max_z*, bool *bReload*)
Preload NPCs By Regions. By default NPCs are loaded on demand when the *ParaTerrain.GetElevation* is called.

Return the number of NPC is returned.

Parameters

- `pOut`: : the list will be filled with data in the database that meat the criterion.
- `vMin`: min point in world coordinate system, y component is ignored
- `vMax`: max point in world coordinate system, y component is ignored
- `bReload`: if the character already exists in the scene and it is true, the character will be updated with the parameter in the database

void **`SetCharacterRegionPath`**(int *slot*, **const** std::string &*path*)
NOTE by andy: 2011.5.23 set character customization system region path setting allow different projects share the same CCS implementation and separate region paths for asset management

Parameters

- `slot`: region slot id
- `path`: new region path NOTE: path string recorded in a static map <slot, path>

void **`SetCharTextureSize`**(int *nCharacterTexSize*, int *nCartoonFaceTexSize*)
the default character skin and cartoon face texture size.

Note : One must call SetCharRegionCoordinates afterwards to change the region accordingly.

Parameters

- `nCharacterTexSize`: the character texture size. default to 256. 512 is also fine.
- `nCartoonFaceTexSize`: the character texture size. default to 256. 128 is also fine.

void **`SetCharRegionCoordinates`**(int *nRegionIndex*, int *xpos*, int *ypos*, int *xsize*, int *ysize*)
set char region coordinates. This function together with SetCharTextureSize makes CCS regions fully customizable via scripting interface. however, the region index are predefined and can not be changed.

Parameters

- `nRegionIndex`: see enumeration CharRegions. it must be smaller than NUM_REGIONS.
- `xposyposxsizeysize`: see struct CharRegionCoords. defines a rect region within the skin or face texture.

void **SetTimeOfDay** (float *time*)
set time of day in seconds. Use *SetDayLength()* to set the total number of minutes in a day.

Parameters

- `time`: time in seconds. This can be any non-negative value. (timeday_length) will be used in case time is longer than a day.

void **SetTimeOfDaySTD** (float *time*)
set standard time. see *SetTimeOfDay()*

Parameters

- `time`: always in the range [-1,1], 0 means at noon, -1 is morning. 1 is night.

float **GetTimeOfDay** ()
get the current time of day in seconds.

float **GetTimeOfDaySTD** ()
get standard time. see *GetTimeOfDay()*

Return : always in the range [-1,1], 0 means at noon, -1 is morning. 1 is night.

void **SetMaximumAngle** (float *fMaxAngle*)
set the maximum sun angle with the Y (up) axis.

float **AdvanceTimeOfDay** (float *timeDelta*)
advance time is in seconds, this will also change the light direction, sun color and sun ambient color. this function is automatically called by the environment simulator

void **SetDayLength** (float *fMinutes*)
set how many minutes are there in a day, this is used in time simulation. default value is 300.

float **GetDayLength** ()
return how many minutes are there in a day in minutes

void **SetShadowMethod** (int *nMethod*)
the current shadow method:both shadow volume and shadow mapping could be set.

Parameters

- `nMethod`: 0: turn off all shadow rendering: this will greatly improve performance 1: turn on shadow using shadow volume 2: turn on shadow mapping 3: turn on both shadow volume and shadow mapping

void **EnableLighting** (bool *bEnable*)
Enable both global and local lighting. Turn off lighting will greatly improve performance, such as on slower computers

void **SetFog** (bool *bRenderFog*, **const** char **strFogColor*, float *fFogStart*, float *fFogEnd*, float *fFogDensity*)
set the global fog effect

Parameters

- `bRenderFog`: 1 to enable fog.
- `strFogColor`: a string of RGB value in the format "%f %f %f", such as "1.0 1.0 1.0", value must be in the range [0, 1.0].
- `fFogDensity`: between (0,1)
- `fFogStart`: unit in meters.

- `fFogEnd`: unit in meters.

ParaScripting::*ParaMiniSceneGraph* **GetMiniSceneGraph** (**const** char **name*)
If the mini scene graph does not exist, it will be created

int **DeleteMiniSceneGraph** (**const** char **name*)
Delete mini scene graphs by name. if name is "" or "*", all mini scene graphs will be deleted.

Return : the number of graphs deleted is returned.

void **EnableMiniSceneGraph** (bool *bEnable*)
enable to render all mini scene graphs

bool **IsMiniSceneGraphEnabled** ()
enable to render all mini scene graphs

ParaScripting::*ParaMiniSceneGraph* **GetPostProcessingScene** ()
get the post processing mini scene graph.

Return : this is same as GetMiniSceneGraph("_ps")

void **EnablePostProcessing** (bool *bEnable*, **const** char **sCallbackScript*)
set whether post processing is enabled.

Parameters

- `bEnable`: bool
- `sCallbackScript`: if bEnabled is true, this is a callback script to be called per frame at which one can process scene after the main 3d scene is rendered. if this is NULL, the old callback script is used.

bool **IsPostProcessingEnabled** ()
get whether post processing is enabled.

bool **GetScreenPosFrom3DPoint** (float *x*, float *y*, float *z*, **const** object &*output*)
Get screen 2D position(x,y) from a 3d world point.

Return true if the point is visible in screen.

Parameters

- `xyz`: in world coordinate system.
- `output`: output table, containing additional info, such as {x=screen position x,y=screen position y,z= depth in the 0,1 range, visible=bool:is point is visible in camera frustum, distance=float:distance from eye to the object, }

void **SetMaxRenderCount** (int *nRenderImportance*, int *nCount*)
set the max number of objects to be rendered of a given render importance. One can set the render importance of any objects by setting its "RenderImportance" property

Parameters

- `nRenderImportance`: the render importance to set. All objects are by default set with render importance 0.
- `nCount`: the max number to set for the above render importance.

int **GetMaxRenderCount** (int *nRenderImportance*)
Get the max number of objects to be rendered of a given render importance. One can set the render importance of any objects by setting its "RenderImportance" property

Parameters

- `nRenderImportance`: the render importance to set. All objects are by default set with render importance 0.
- `nCount`: the max number to set for the above render importance.

class **`ParaSearchResult`**

#include <ParaScriptingIO.h> it represents a search result.

Public Functions

bool **`IsValid`**()

check if the object is valid

void **`Release`**()

release results.

string **`GetRootPath`**()

get the root path

int **`GetNumOfResult`**()

get the total number of result found

bool **`AddResult`**(**const** char **sItem*)

add a new item. return false if the search is completed.

string **`GetItem`**(int *nIndex*)

get the item at the specified index. return "" if nIndex is out of range.

object **`GetItemData`**(int *nIndex*, **const** object *&output*)

get the item at the specified index. return nil if nIndex is out of range.

Return : return the field result. If field not found, output will be returned. e.g. {filename="abc.txt", filesize=0, fileattr=0, createdate="1982-11-26", writedate="", accessdate=""} fileattr is the value same as defined in WIN32_FIND_DATA.dwFileAttributes

Parameters

- `output`: [in|out] default value.

class **`ParaSelection`**

#include <ParaScriptingWorld.h> A pool of currently selected objects. This is a singleton class. Object may be selected into different groups. Although, there are no limit to group number, better keep it smaller than 16 groups. Selected objected may be displayed or highlighted differently. When objects are deleted from the scene. It will be deleted from the selection automatically.

Public Static Functions

void **`RemoveObject`**(**const** ParaObject *&obj*)

remove a given object in all selections.

Parameters

- `obj`: pointer to the object to delete

bool **`RemoveUIObject`**(**const** char **sName*)

remove an UI object from all selection groups.

Return if if removed

Parameters

- `sName`: UI object name

void **`AddObject`** (**const** ParaObject &*obj*, int *nGroupID*)
Add a new object to a given group. An object may belong to multiple groups.

Parameters

- `obj`: pointer to the object to add
- `nGroupID`: which group the should be added to. by default it is added to group 0. group ID must be smaller than 32.

bool **`AddUIObject`** (**const** char **sName*, int *nGroupID*)
Please note that UI object in selection is automatically highlighted using the default highlighting effect.

Return true if the object is found and highlighted(selected).

Parameters

- `sName`: UI object name
- `nGroupID`: which group the should be added to. by default it is added to group 0.

ParaObject **`GetObject`** (int *nGroupID*, int *nItemIndex*)
get the nItemIndex object in the nGroupID group.

Return selected item is returned

Parameters

- `nGroupID`: from which group the object is get
- `nItemIndex`: the index of the item to be retrieved.

void **`GetObject_`** (ParaObject **pOut*, int *nGroupID*, int *nItemIndex*)
this function shall never be called from the scripting interface. this is solely for exporting API. and should not be used from the scripting interface.

int **`GetItemNumInGroup`** (int *nGroupID*)
get the total number item in the given group. This function can be used with *GetObject()* to iterate through all objects in any group.

Return

Parameters

- `nGroupID`: group ID.

void **`SelectGroup`** (int *nGroupID*, bool *bSelect*)
select the entire group.

Parameters

- `nGroupID`:
- `bSelect`: true to select, false to de-select.

void **`ClearGroup`** (int *nGroupID*)
Clear a given group so that there are no objects in it.

Parameters

- `nGroupID`: ID of the group. If ID is -1, all groups will be deleted.

void **SetMaxItemNumberInGroup**(int *nGroupID*, int *nMaxItemsNumber*)
set the maximum number of objects in the group.

Parameters

- `nGroupID`: group ID group ID must be smaller than 32.
- `nMaxItemsNumber`: the number to set. default value is 1

int **GetMaxItemNumberInGroup**(int *nGroupID*)
set the maximum number of objects in the group.

Return the maximum number in the given group

Parameters

- `nGroupID`: group ID,which ID must be smaller than 32.

class **ParaSeqCtrler**
#include <ParaScriptingCharacter.h> ParaSeqCtrler object: A sequence controller is a biped controller which moves the biped according to some predefined sequence. it is used to control sequence based commands of character object.

Public Functions

bool **IsValid**()
check if the object is valid

int **GetKeyPos**()
Get the current absolute playing cursor position

void **SetKeyPos**(int *nPos*)
set the current absolute playing cursor position

int **GetTotalKeys**()
get total key count

int **AdvanceKey**(int *nOffset*)
offset the key index according to the current play mode. i.e. it will automatically wrap to the beginning if looping.

Return : the number of keys that have been successfully offseted. Usually if the returned value is not equal to the input value, it means that the sequence should be paused.

Parameters

- `nOffset`: number of keys to advance.

void **BeginAddKeys**()
call the command functions(RunTo, MoveTo, etc) only between the matching pair of *BeginAddKeys()* and *EndAddKeys()*

void **EndAddKeys**()
call the command functions(RunTo, MoveTo, etc) only between the matching pair of *BeginAddKeys()* and *EndAddKeys()*

bool **DeleteKeysRange**(int *nFrom*, int *nTo*)
delete keys range

Parameters

- `nFrom`: 0 based index.
- `nTo`: 0 based index, if -1, it means the last one.

bool **`GetPlayDirection`**()
get the play direction.

void **`SetPlayDirection`**(bool *bForward*)
set the play direction.

float **`GetInterval`**()
the minimum time between two successive calls.

void **`SetInterval`**(float *fInterval*)
the minimum time between two successive calls.

void **`GetStartPos`**(float &*x*, float &*y*, float &*z*)
get the starting position.

void **`SetStartPos`**(float *x*, float *y*, float *z*)
set the starting position.

float **`GetStartFacing`**()
get the start facing. usually default to 0.

void **`SetStartFacing`**(float *facing*)
Set the start facing. usually default to 0.

int **`GetPlayMode`**()
get the current play mode

void **`SetPlayMode`**(int *mode*)
set the current play mode

float **`GetMovingTimeout`**()
get the number of seconds after which all move commands will be treated as finished. default value is 30 seconds.

void **`SetMovingTimeout`**(float *fTimeout*)
set the number of seconds after which all move commands will be treated as finished. default value is 30 seconds.

void **`Suspend`**()
suspend

void **`Resume`**()
resume

class `ParaServiceLogger`
#include <ParaScriptingGlobal.h> service logger

Public Functions

void **`log`**(int *level*, **const** object &*message*)
log to the current logger

int **GetLevel**()
Returns the assigned Level

Return Level - the assigned Level

void **SetLevel**(**const** int *level1*)
set level of this logger.

bool **IsEnabledFor**(int *level*)
Check whether this logger is enabled for a given Level passed as parameter.

Return bool True if this logger is enabled for level. It just checks (level>=this->m_level)

void **SetAppendMode**(bool *bAppendToExistingFile*)
by default, append mode is enabled in server mode, and disabled in client build.

Note : only call this function when no log is written before using the logger.

void **SetLogFile**(**const** char **sFileName*)
change the log file.

Note : only call this function when no log is written before using the logger.

Parameters

- `sFileName`: such as "log/test.log"

void **SetForceFlush**(bool *bForceFlush*)
if true we will flush the new log to file immediately. otherwise, flush operation is determined by the system. default to true for client log and false for service log.

Note : only call this function when no log is written before using the logger.

class ParaTerrain
#include <ParaScriptingTerrain.h> Contains Terrain functions

Public Static Functions

ParaAttributeObject **GetAttributeObject**()
get the attribute object associated with the global terrain.

void **GetAttributeObject_**(*ParaAttributeObject &output*)
used for API exportation.

ParaAttributeObject **GetBlockAttributeObject**()
get block terrain manager's attribute object.

int **GetTextureCount**(float *x*, float *y*)
get the total number of textures in the texture set of the terrain tile specified by a world position (x,y) One can get each of the textures by calling *GetTexture()* function.

void **GetTexturesInCell**(float *x*, float *y*, **const** object *&out*)
get all texture indices in the cell specified by point(x,y) *GetTextureCount()* returns all textures used by the entire terrain tile (i.e. 500*500). however, each terrain tile is further subdevided into 8*8=64 terrain cell (each cell takes up about 64*64 meters). Alpha channels of a terrain texture is only created on a per cell basis. A single alpha image (128*128) will be created for each used texture in the cell.

Parameters

- `x`: in world unit
- `y`: in world unit
- `out`: this should be an empty table to be filled with textures in the cell, so that {[1]=tex_index, [2]=tex_index, ...}.

bool **`RemoveTextureInCell`** (float *x*, float *y*, int *nIndex*)
remove the given texture in the cell specified by the point(x,y).

Parameters

- `x`: in world unit
- `y`: in world unit
- `nIndex`: Texture index in the current terrain tile. this should be smaller than the total number of textures in the texture set. see *GetTextureCount()*.

ParaAssetObject **`GetTexture`** (float *x*, float *y*, int *nIndex*)
get the terrain detailed texture by index. Please note that each terrain tile contains its own texture set. the total number of textures can be returned by *GetTextureCount()*.

Return : The texture entity object is returned. The returned object may be invalid if nIndex is invalid.

Parameters

- `nIndex`: Texture index in the current terrain tile. this should be smaller than the total number of textures in the texture set. see *GetTextureCount()*. there are two reserved negative index for the common and main textures -1: common texture, which is repeated several times over each terrain tile surface. -2: main texture, which is chopped and mapped to the entire terrain surface.

void **`GetTexture__`** (*ParaAssetObject* **pOut*, float *x*, float *y*, int *nIndex*)
only used for API exporting.

bool **`ReplaceTexture`** (float *x*, float *y*, int *nIndex*, *ParaAssetObject* &*TextureAsset*)
replace the texture at the given index. or append a new texture, or delete a texture if TextureAsset or sTextureAsset is NULL.

Parameters

- `nIndex`: if it is smaller than the total number of textures in the texture set. see *GetTextureCount()*. there are two reserved negative index for the common and main textures -1: common texture, which is repeated several times over each terrain tile surface. -2: main texture, which is chopped and mapped to the entire terrain surface.
- `sTextureAsset`: filename. if nil, texture will be removed.

ParaAttributeObject **`GetAttributeObjectAt`** (float *x*, float *y*)
get the attribute object associated with the terrain tile that contains the given point.

Return

Parameters

- `x`: in world unit
- `y`: in world unit

void **`GetAttributeObjectAt_`** (*ParaAttributeObject* &*output*, float *x*, float *y*)
used for API exportation.

float **`GetElevation`** (float *x*, float *y*)
get the terrain height at the specified position.

Return : the terrain height.

Parameters

- `x`: position in world unit
- `y`: position in world unit

DWORD **`GetRegionValue`** (**const** char **sLayerName*, float *x*, float *y*)
get value of a given terrain region layer

Parameters

- `sLayerName`: the layer name
- `x`: The x location of the point on the Terrain's surface in world units.
- `y`: The y location of the point on the Terrain's surface in world units.

DWORD **`GetRegionValue4`** (**const** char **sLayerName*, float *x*, float *y*, **const** char **argb*)
get value of a given terrain region layer

Parameters

- `sLayerName`: the layer name
- `x`: The x location of the point on the Terrain's surface in world units.
- `y`: The y location of the point on the Terrain's surface in world units.
- `argb`: it can be a string containing "argb", where the sum of them are returned. for example, specify "r" to get only the red channel value. specify "rgb" for the sum of the grey scale image. if this is empty string "", the 32bits color value is returned as int.

string **`GetTerrainOnloadScript`** (float *x*, float *y*)
get the on load script which created all objects on this terrain that contains the point (x,y)

const char ***`GetTerrainOnloadScript__`** (float *x*, float *y*)
solely used for API exporting. Not thread-safe

string **`GetTerrainElevFile`** (float *x*, float *y*)
get the height map file name for the terrain tile that contains the point (x,y)

const char ***`GetTerrainElevFile__`** (float *x*, float *y*)
solely used for API exporting. Not thread-safe

void **`SaveTerrain`** (bool *bHeightMap*, bool *bTextures*)
save modified terrain to disk.

Parameters

- `bHeightMap`: true to save height map
- `bTextures`: true to save textures: alpha maps, etc

void **`ReloadTerrain`** (bool *bHeightMap*, bool *bTextures*)
reload terrain from disk. User will lose changes since last save operation. one can UNDO changes with this function.

Parameters

- `bHeightMap`: true to save height map

- `bTextures`: true to save textures: alpha maps, etc

void **`UpdateTerrain`**()

update terrain. this function is called, when the user changes the terrain surface.

Parameters

- `bForceUpdate`: if true it will update the terrain even if the camera does not moves.

void **`SnapPointToVertexGrid`**(float *x*, float *y*, float **vertex_x*, float **vertex_y*)

snap any 2D point on the height map to a vertex position on the height map. in NPL, one should write as below: local x,y = ParaTerrain.SnapPointToVertexGrid(10,10);

Parameters

- `x`: arbitrary 2D point on the height map
- `y`: arbitrary 2D point on the height map
- `vertex_x`: [out] vertex position on the height map
- `vertex_y`: [out] vertex position on the height map

float **`GetVertexSpacing`**()

Returns the number of real units between vertices in the terrain's mesh.

void **`Paint`**(**const** char **detailTexture*, float *brushRadius*, float *brushIntensity*, float *maxIntensity*, bool *erase*, float *x*, float *y*)

paint on the specified location of the global terrain.

Parameters

- `detailTexture`: the texture entity to paint on the terrain. The texture is usually tilable such as road and grass. if this is "", it means the base layer or layer 0. Since version 0.9.9, the base layer also has an editable alpha mask .
- `brushRadius`: The width of the brush to paint with in DetailTexture layer pixels. There are typically 256 of these pixels across a TextureCell.
- `brushIntensity`: The intensity with which to paint, ranging from 0.0 to 1.0. This determines how opaque the painted "splat" will be.
- `maxIntensity`: The maximum intensity of the painted splat, ranging from 0.0 to 1.0, accounting for mixing with a splat that may already reside on the surface.
- `erase`: Specifies whether to add the splat to the surface or remove existing splat texels already on the surface (pass false to paint and pass true to erase existing splats)
- `x`: The x location of the point to paint on the Terrain's surface in world units.
- `y`: The y location of the point to paint on the Terrain's surface in world units.

void **`Paint_`**(int *nDetailTextureID*, float *brushRadius*, float *brushIntensity*, float *maxIntensity*, bool *erase*, float *x*, float *y*)

See *Paint()*.

Parameters

- `nDetailTextureID`: multi-texture layer ID, beginning from 0. Such as 0,1,2,3. -1,-2 are reserved for special terrain editor textures. -1 means the base layer.

void **`DigCircleFlat`**(float *x*, float *y*, float *radius*, float *fFlatPercentage*, float *factor*)

flatten a land centered at x,y, with a specified radius. Algorithm: (1) flatten a circle with radius same as fFlatPercentage*radius (2) smooth the entire region twice.

Parameters

- `x`: center of the circle in world unit
- `y`: center of the circle in world unit
- `radius`: radius of the inner circle in world unit
- `fFlatPercentage`: value is between [0,1]. fFlatPercentage*radius is the actual radius that got flattened.
- `factor`: value is between [0,1]. 1 means fully transformed; 0 means nothing is changed

void **`Flatten`** (float *x*, float *y*, float *radius*, int *flatten_op*, float *elevation*, float *factor*)

Flatten the terrain both up and down to the specified elevation, using using the tightness parameter to determine how much the altered points are allowed to deviate from the specified elevation.

Parameters

- `x`: center of the circle in world unit
- `y`: center of the circle in world unit
- `radius`: radius of the inner circle in world unit
- `flatten_op`: enum FlattenOperation{ Fill_Op=0, //Flatten the terrain up to the specified elevation ShaveTop_Op=1, //Flatten the terrain down to the specified elevation Flatten_Op=2, //Flatten the terrain up and down to the specified elevation };
- `elevation`: the desired height
- `factor`: value is between [0,1]. 1 means fully transformed; 0 means nothing is changed

void **`RadialScale`** (float *x*, float *y*, float *scale_factor*, float *min_dist*, float *max_dist*, float *smooth_factor*)

Note: terrain data should be in normalized space with height in the range [0,1]. Picks a point and scales the surrounding terrain in a circular manner. Can be used to make all sorts of circular shapes. Still needs some work. radial_scale: pick a point (center_x, center_y) and scale the points where distance is mindist<=distance<=maxdist linearly. The formula we'll use for a nice sloping smoothing factor is (-cos(x*3)/2)+0.5.

Parameters

- `x`: center of the circle in world unit
- `y`: center of the circle in world unit
- `scale_factor`: height of the scaled portion in world unit. This value can be negative.

void **`Spherical`** (float *x*, float *y*, float *radius*, float *offset*)

offset in a spherical region

Parameters

- `x`: center of the circle in world unit
- `y`: center of the circle in world unit
- `radius`: radius of the inner circle in world unit

void **`Roughen_Smooth`** (float *x*, float *y*, float *radius*, bool *roughen*, bool *big_grid*, float *factor*)

square filter for sharpening and smoothing. Use neighbour-averaging to roughen or smooth the height field. The factor determines how much of the computed roughening is actually applied to the height field. In it's default invocation, the 4 directly neighboring squares are used to calculate the roughening. If you select big sampling grid, all 8 neighboring cells will be used.

Parameters

- `x`: center of the circle in world unit
- `y`: center of the circle in world unit
- `radius`: radius of the inner circle in world unit
- `roughen`: true for sharpening, false for smoothing.
- `big_grid`: true for 8 neighboring cells, false for 4.
- `factor`: value is between [0,1]. 1 means fully transformed; 0 means nothing is changed

void **`Ramp`** (float *x1*, float *y1*, float *x2*, float *y2*, float *radius*, float *borderpercentage*, float *factor*)

create a ramp (inclined slope) from (x1,y1) to (x2,y2). The ramp's half width is radius. this is usually used to created a slope path connecting a high land with a low land.

Parameters

- `radius`: The ramp's half width
- `borderpercentage`: borderpercentage*radius is how long the ramp boarder is to linearly interpolate with the original terrain. specify 0 for sharp ramp border.
- `factor`: in range[0,1]. it is the smoothness to merge with other border heights.Specify 1.0 for a complete merge

void **`AddHeightField`** (float *x*, float *y*, **const** char **filename*, int *nSmoothPixels*)

Add rectangular height field from file to the current terrain.

Parameters

- `x`: center of the rect in world unit
- `y`: center of the rect in world unit
- `filename`: : the raw elevation or gray scale image file that contains the height field.
- `nSmoothPixels`: the number of pixels to smooth from the edge of the height field. if this is 0, the original height field will be loaded unmodified. if it is greater than 0, the loaded height field will be smoothed for nSmoothPixels from the edge, where the edge is always 0.

void **`MergeHeightField`** (float *x*, float *y*, **const** char **filename*, int *mergeOperation* = 0, float *weight1* = 1.0, float *weight2* = 1.0, int *nSmoothPixels* = 7)

merge a rectangular height field from file to the current terrain.

Parameters

- `x`: center of the rect in world unit
- `y`: center of the rect in world unit
- `filename`: : the raw elevation or gray scale image file that contains the height field.
- `MergeOperation`: { Addition=0, Subtract=1, Multiplication=2, Division=3, Minimum=4, Maximum=5, };
- `weight1`: the destination merging weight
- `weight2`: the source file merging weight
- `nSmoothPixels`: the number of pixels to smooth from the edge of the height field. if this is 0, the original height field will be loaded unmodified. if it is greater than 0, the loaded height field will be smoothed for nSmoothPixels from the edge, where the edge is always 0.

void **UpdateHoles** (float *x*, float *y*)
Update all holes in the terrain tile that contains the input point.

Parameters

- `x`: The x location of the point on the Terrain's surface in world units.
- `y`: The y location of the point on the Terrain's surface in world units.

bool **IsHole** (float *x*, float *y*)
Whether the terrain contains a hole at the specified location. Currently, we allow user to load a low resolution hole maps at the beginning of terrain creation.

Return : true if the position specified by (x,y) is inside a terrain hole

Parameters

- `x`: The x location of the point on the Terrain's surface in world units.
- `y`: The y location of the point on the Terrain's surface in world units.

void **SetHole** (float *x*, float *y*, bool *bIsHold*)
set a new terrain hole at the specified location. Currently, we will allow user to dynamically dig holes in terrain. After calling this function, the user must manually Call *UpdateHoles()* to inform that the holes in the terrain has been updated.

See *UpdateHoles()*;

Parameters

- `x`: The x location of the point on the Terrain's surface in world units.
- `y`: The y location of the point on the Terrain's surface in world units.

bool **IsModified** ()
return true, if the terrain is modified and should be saved.

void **SetContentModified** (float *x*, float *y*, bool *bIsModified*)
set the content of the terrain modified. the terrain is specified by a 2D point. the on load script will be rebuilt once saving the terrain.

Parameters

- `x`: a position on the terrain where content is changed.
- `y`: a position on the terrain where content is changed.
- `bIsModified`: true to set modified.

void **SetContentModified4** (float *x*, float *y*, bool *bIsModified*, DWORD *dwModifiedBits*)
set the content of the terrain modified. the terrain is specified by a 2D point. the on load script will be rebuilt once saving the terrain.

Parameters

- `bIsModified`: true to set modified.
- `dwModifiedBits`: this is any combination of TERRAIN_MODIFIED_BITS. Default value is MODIFIED_ON_LOAD_SCRIPT (16) enum TERRAIN_MODIFIED_BITS { MODIFIED_NONE = 0, detailed terrain texture(with mask) has been modified. MODIFIED_TEXTURE = 0x1, height map has modified MODIFIED_HEIGHTMAP = 0x1<<1, configuration such as base texture, common file, holes, etc has been modified. MODIFIED_CONFIGURATION = 0x1<<2, holes have been changed. this should mean the same thing as MODIFIED_CONFIGURATION MODIFIED_HOLES = 0x1<<3, if static

objects have been modified, so that we will need to update the on load script MODIFIED_ON_LOAD_SCRIPT = 0x1<<4, MODIFIED_ALL = 0xffff };

void **SetAllLoadedModified**(bool *bIsModified*, DWORD *dwModifiedBits*)
set all loaded terrain tile content modified. This is the refered way to perform a save all operation

void **EnableLighting**(bool *bEnable*)
Enable both global and local lighting. Turn off lighting will greatly improve performance, such as on slower computers

bool **RegisterBlockTemplate**(uint16_t *templateId*, **const** object &*params*)

Parameters

- `params`: it can be attFlag of int type.

object **GetBlocksInRegion**(int32_t *startChunkX*, int32_t *startChunkY*, int32_t *startChunkZ*, int32_t *endChunkX*, int32_t *endChunkY*, int32_t *endChunkZ*, uint32_t *matchType*, **const** object &*result*)
get block in [startChunk,endChunk]

Return {count,x{},y{},z{},tempId{}}

Parameters

- `startChunkYendChunkY`: if negative, and startChunkY == endChunkY, -startChunkY will be used as verticalSectionFilter(a bitwise filter).

void **SelectBlock**(uint16_t *x*, uint16_t *y*, uint16_t *z*, bool *isSelect*)
add/remove block to/from highlight block list

Parameters

- `xyz`: world space block id;
- `isSelect`: : true to select block, false to de-select block
- `nGroupID`: group id. 0 for highlight 1 for wireframe.

void **DeselectAllBlock1**(int *nGroupID*)

Parameters

- `nGroupID`: 0 for animated selection, 1 for wire frame selection. -1 for all

int **FindFirstBlock**(uint16_t *x*, uint16_t *y*, uint16_t *z*, uint16_t *nSide* = 4, uint32_t *max_dist* = 32, uint32_t *attrFilter* = 0xffffffff, int *nCategoryID* = -1)
find a block in the side direction that matched filter from block(x,y,z) this function can be used to check for free space upward or download

Return -1 if not found. otherwise distance to the first block that match in the side direction is returned.

Parameters

- `side`: 4 is top. 5 is bottom.
- `attrFilter`: attribute to match. 0 means air. default to any block
- `nCategoryID`: -1 means any category_id. default to -1

int **GetFirstBlock**(uint16_t *x*, uint16_t *y*, uint16_t *z*, int *nBlockId*, uint16_t *nSide* = 4, uint32_t *max_dist* = 32)
get the y pos of the first block of nBlockID, start searching from x, y, z in the side direction

int32_t **`GetChunkColumnTimeStamp`**(uint32_t *chunkX*, uint32_t *chunkZ*)
get the time stamp of for the given chunk column 0 means not available, >1 means loaded before

void **`SetChunkColumnTimeStamp`**(uint32_t *chunkX*, uint32_t *chunkZ*, uint32_t *nTimeStamp*)
set chunk column time stamp. usually 0 for non-generated. 1 for generated. this is usually called by world generators, so that we will not generate again next time we load the world.

const std::string &**`GetMapChunkData`**(uint32_t *chunkX*, uint32_t *chunkZ*, bool *bIncludeInit*, uint32_t *verticalSectionFilter*)

Parameters

- `verticalSectionFilter`: default to 0xffff. each bit is for one of the 16 vertical sections.

class `ParaUI`
#include <ParaScriptingGUI.h> *ParaUI* namespace contains a list of HAPI functions to create user interface controls, such as windows, buttons, as well as event triggers. The main difference between the two is that (1) 2D GUI object are not tested against view frustum, instead its controlled by visibility tag automatically or through user input. (2) 2D GUI object generally does not obey the physical law of 3D world. (3) GUI object are generally specified by screen coordinates, instead of 3D position. (4) GUI object may be frequently created and discarded. If a object is discarded, so will its child objects. E.g. if one delete a window, all buttons, sensors, etc attach to it will also be discarded.

Public Static Functions

void **`PlaySound`**(**const** char **strSoundAssetName*, bool *bLoop*)
Play a sound file

Parameters

- `strObjectName`: the sound object name

void **`StopSound`**(**const** char **strSoundAssetName*)
stop playing a sound file. one can use this function to stop playing looping sound such as a background music.

Parameters

- `strObjectName`: the sound object name

void **`Destroy`**(**const** char **strObjectName*)
delete a GUI object as well as all its child objects, by it name. If there are several objects who have the same id, only the last attached one is deleted.

Parameters

- `strObjectName`: the object name

void **`Destroy1`**(int *nID*)
delete a GUI object as well as all its child objects, by it name. If there are several objects who have the same id, only the last attached one is deleted.

Parameters

- `nID`: id.

void **`DestroyUIObject`**(*ParaUIObject* &*obj*)
destroy an UI object

void **PostDestroy** (**const** char **strObjectName*)
delete at the end of frame.

ParaUIObject **GetUIObject_any** (**const** object &*NameOrID*)
Get GUI object by name or ID

Parameters

- `NameOrID`: if it is string, we will get the first GUI object that matches the name. If name is "root", the root object is returned. if it is number , we will get the object by its ID. NameOrID is then the id property value or result of GetID() from a UI object. If this is 0, the root object is returned.

ParaUIObject **GetUIObjectAtPoint** (int *x*, int *y*)
Get the first GUI object at the given coordinates x: x coordinate y: y coordinate

void **GetMousePosition** (float &*x*, float &*y*)
get the current mouse position. e.g. local x,y = *ParaUI.GetMousePosition()*;

Parameters

- `&x`: screen x
- `&y`: screen y

ParaUIObject **CreateUIObject** (**const** char **strType*, **const** char **strObjectName*, **const** char **alignment*, int *x*, int *y*, int *width*, int *height*)
Create a GUI object based on the default template object. All default template object is defined in "script/config.lua", which will be loaded when GUI engine is loaded. One can change template object at runtime by *GetDefaultObject()*. the layout is given below: _lt _mt _rt _ml _ct _mr _lb _mb _rb

See *GetDefaultObject()* Although it is possible to create many objects with the same name, we do not advice you to do so.

Remark : we design _m? alignment types. _mt: x is coordinate from the left. y is coordinate from the top, width is the coordinate from the right and height is the height _mb: x is coordinate from the left. y is coordinate from the bottom, width is the coordinate from the right and height is the height _ml: x is coordinate from the left. y is coordinate from the top, width is the width and height is the coordinate from the bottom _mr: x is coordinate from the right. y is coordinate from the top, width is the width and height is the coordinate from the bottom

Parameters

- `strType`: type of the new object. It can be "container", "button", "scrollbar", "editbox", "imeeditbox","slider", "video", "3dcanvas", "listbox", "painter" and "text". "container" is the only type of control that can contain other objects
- `strObjectName`: the new object's name
- `alignment`: can be one of the following strings or nil or left out entirely:
  - "_lt" align to left top of the screen
  - "_lb" align to left bottom of the screen
  - "_ct" align to center of the screen
  - "_ctt": align to center top of the screen
  - "_ctb": align to center bottom of the screen
  - "_ctl": align to center left of the screen

- "_ctr": align to center right of the screen
- "_rt" align to right top of the screen
- "_rb" align to right bottom of the screen
- "_mt": align to middle top
- "_ml": align to middle left
- "_mr": align to middle right
- "_mb": align to middle bottom
- "_fi": align to left top and right bottom. This is like fill in the parent window.

Parameters

- `x`: screen coordinate x, for _m? alignments, the meaning is different, see remark
- `y`: screen coordinate y, for _m? alignments, the meaning is different, see remark
- `width`: screen coordinate width or right, depending on alignment mode, for _m? alignments, the meaning is different, see remark
- `height`: screen coordinate height or bottom, depending on alignment mode, for _m? alignments, the meaning is different, see remark

ParaScripting::*ParaUIObject* **`GetTopLevelControl`**()

get the top level control at level 0

ParaUIObject **`GetDefaultObject`**(**const** char **strType*)

get the default template object from which all sub-sequent controls of the same type are cloned(created). one can modify the template object at runtime to change of the theme of all controls created subsequently. All default template object is defined in "script/config.lua", which will be loaded when GUI engine is loaded

Return the template object is returned.

Parameters

- `strType`: It can be "container", "button", "scrollbar", "editbox", "imeeditbox","slider" and "text".

void **`SetCursorFont`**(**const** char **fontname*, **const** char **strColor*, DWORD *transparency*)

@ obsoleted. Set Mouse Cursor appearance

void **`SetCursorTexture`**(**const** char **texturename*, **const** char **strColor*, DWORD *transparency*)

. Set Mouse Cursor appearance

void **`SetCursorText`**(**const** char **strText*)

. Set Mouse Cursor appearance

void **`SetCursorFromFile`**(**const** char **szCursor*, int *XHotSpot*, int *YHotSpot*)

Set the current cursor to use. One can call very often, since it will does nothing with identical cursor file and hot spot. Typically, hardware supports only 32x32 cursors and, when windowed, the system might support only 32x32 cursors.

Parameters

- `szCursor`: cursor file name: The contents of this texture will be copied and potentially format-converted into an internal buffer from which the cursor is displayed. The dimensions of this surface must be less than the dimensions of the display mode, and must be a power of two in each direction, although not necessarily the same power of two. The alpha channel must be either 0.0 or 1.0.

- `XHotSpot`: [in] X-coordinate offset (in pixels) that marks the center of the cursor. The offset is relative to the upper-left corner of the cursor. When the cursor is given a new position, the image is drawn at an offset from this new position determined by subtracting the hot spot coordinates from the position.
- `YHotSpot`: [in] Y-coordinate offset (in pixels) that marks the center of the cursor. The offset is relative to the upper-left corner of the cursor. When the cursor is given a new position, the image is drawn at an offset from this new position determined by subtracting the hot spot coordinates from the position.

void **SetCursorFromFile_**(**const** char **szCursor*)
same as above, with hot spot (0,0)

void **SetUseSystemCursor**(bool *bUseSystem*)
whether to use system cursor. If this is true, d3d hardware cursor is not shown, even you have loaded an cursor icon using SetCursorFromFile.

bool **GetUseSystemCursor**()
get whether to use system cursor

string **ToScript**()
to NPL script.

bool **SaveLayout**(**const** char **filename*)
save current layout to file

Return

Parameters

- `*filename`:

void **SetDesignTime**(bool *bDesign*)
whether to enable edit mode for all controls.

Parameters

- `bDesign`:

void **ShowCursor**(bool *bShow*)
show cursor

Parameters

- `bShow`:

void **LockMouse**(bool *bLock*)
Lock Mouse so that mouse move will not change the mouse position. this is useful when user is changing camera view during mouse drag operation.

Parameters

- `bLock`: true to lock

bool **IsMouseLocked**()
check whether Mouse is locked. When mouse is locked, mouse move will not change the mouse position. this is useful when user is changing camera view during mouse drag operation.

void **ResetUI**()
clear all UI objects.

void **SetIMEOpenStatus**(bool *bOpen*)
This function opens or closes the IME programmtically. In most cases, we should never programmatically open or closes IME, instead the user usually pressed Ctrl+Space to change it. however, in some rare cases, such as we are opening a windowed mode flash window, and wants to disable IME programmatically.

Parameters

- `bOpen`: true to open.

bool **GetIMEOpenStatus**()
Check if IME status is opened.

void **SetUIScale**(float *fScalingX*, float *fScalingY*)
set the UI scaling. This can be useful to render 1024*768 to a 800*600 surface; we can set to fScalingX to 800/1024 and fScalingY to 600/768 calling this function will cause OnSize() and UpdateBackbufferSize() to be called.

Parameters

- `fScalingX`: x defaults to 1.0
- `fScalingY`: y defaults to 1.0

void **SetMinimumScreenSize**(int *nWidth*, int *nHeight*, bool *bAutoUIScaling*)
the minimum screen size. if the backbuffer is smaller than this, we will use automatically use UI scaling for example, if minimum width is 1024, and backbuffer it 800, then m_fUIScalingX will be automatically set to 1024/800.

Parameters

- `nWidth`: the new width.
- `nHeight`: the new height.
- `bAutoUIScaling`: usually set to true. whether we will automatically recalculate the UI scaling accordingly with regard to current backbuffer size.

void **AddDragReceiver**(**const** char **sName*)
add an receiver to the current receiver list during an drag operation. call this function on an dragable UI object's Begin Drag event handler.

Parameters

- `sName`: name. if this is "root", the dragging object can always to reattached.

void **SetToolTipBehavior**(**const** char **behavior*)
How tooltip is displayed

Parameters

- `behavior`: "normal" or "flashing". default is "normal"

bool **SetHighlightParam**(**const** char **szEffectName*, **const** char **szParamName*, **const** char **szParamValue*)
Set Highlight Param. this is usually called in the start up configuration file.

Return

Parameters

- `szEffectName`:
- `szParamName`:
- `szParamValue`:

class `ParaUIFont`

#include <ParaScriptingGraphics.h> a GUI font object

Class Properties

- ("transparency",&ParaUIFont::GetTransparency,&ParaUIFont::SetTransparency)
- ("color",&ParaUIFont::GetColor,&ParaUIFont::SetColor)
- ("font",&ParaUIFont::GetFont,&ParaUIFont::SetFont)
- ("format",&ParaUIFont::GetFormat,&*ParaUIFont::SetFormat*),

Public Functions

bool **`IsValid`**()

check if the object is valid

void **`SetFormat`**(DWORD *format*)

Set the text align and other text displaying formats

Parameters

- `dwFormat`: the new format. It can be any combination of the following values. DT_BOTTOM (0x00000008) Justifies the text to the bottom of the rectangle. This value must be combined with DT_SINGLELINE. DT_CALCRECT (0x00000400) Determines the width and height of the rectangle. If there are multiple lines of text, ID3DXFont::DrawText uses the width of the rectangle pointed to by the pRect parameter and extends the base of the rectangle to bound the last line of text. If there is only one line of text, ID3DXFont::DrawText modifies the right side of the rectangle so that it bounds the last character in the line. In either case, ID3DXFont::DrawText returns the height of the formatted text but does not draw the text. DT_CENTER (0x00000001) Centers text horizontally in the rectangle. DT_EXPANDTABS (0x00000040) Expands tab characters. The default number of characters per tab is eight. DT_LEFT (0x00000000) Aligns text to the left. DT_NOCLIP (0x00000100) Draws without clipping. ID3DXFont::DrawText is somewhat faster when DT_NOCLIP is used. DT_RIGHT (0x00000002) Aligns text to the right. DT_RTLREADING Displays text in right-to-left reading order for bi-directional text when a Hebrew or Arabic font is selected. The default reading order for all text is left-to-right. DT_SINGLELINE (0x00000020) Displays text on a single line only. Carriage returns and line feeds do not break the line. DT_TOP (0x00000000) Top-justifies text. DT_VCENTER (0x00000004) Centers text vertically (single line only). DT_WORDBREAK (0x00000010) Breaks words. Lines are automatically broken between words if a word would extend past the edge of the rectangle specified by the pRect parameter. A carriage return/line feed sequence also breaks the line.

class `ParaUIObject`

#include <ParaScriptingGUI.h> it represents a GUI object.

•("text",&*ParaUIObject::GetText*,&ParaUIObject::SetText1)

•("id",&*ParaUIObject::GetID*,&*ParaUIObject::SetID*)

•("PasswordChar",&*ParaUIObject::GetPasswordChar*,&*ParaUIObject::SetPasswordChar*)

- (“name”,&*ParaUIObject::GetName*,&ParaUIObject::SetName)
- (“enabled”,&ParaUIObject::GetEnabled,&*ParaUIObject::SetEnabled*)
- (“highlightstyle”,&ParaUIObject::GetHighlightStyle,&ParaUIObject::SetHighlightStyle)
- (“autosize”,&ParaUIObject::GetAutoSize,&*ParaUIObject::SetAutoSize*)
- (“visible”,&ParaUIObject::GetVisible,&*ParaUIObject::SetVisible*)
- (“candrag”,&ParaUIObject::GetCanDrag,&ParaUIObject::SetCanDrag)
- (“scrollable”,&ParaUIObject::GetScrollable,&*ParaUIObject::SetScrollable*)
- (“readonly”,&ParaUIObject::GetReadOnly,&ParaUIObject::SetReadOnly)
- (“position”,&*ParaUIObject::GetPosition*,&*ParaUIObject::SetPosition*)
- (“parent”,&ParaUIObject::GetParent,&ParaUIObject::SetParent)
- (“background”,&ParaUIObject::GetBGImage,&*ParaUIObject::SetBGImage1*)
- (“color”,&*ParaUIObject::GetColor*,&*ParaUIObject::SetColor*)
- (“button”,&ParaUIObject::GetBtnImage,&ParaUIObject::SetBtnImage1)
- (“font”,&ParaUIObject::GetFontString,&ParaUIObject::SetFontString1)
- (“type”,&ParaUIObject::GetType)
- (“shadow”,&ParaUIObject::GetUseTextShadow,&*ParaUIObject::SetUseTextShadow*)
- (“textscale”,&*ParaUIObject::GetTextScale*,&*ParaUIObject::SetTextScale*)
- (“script”,&ParaUIObject::ToScript)
- (“ismodified”,&ParaUIObject::IsModified)
- (“animstyle”,&*ParaUIObject::GetAnimationStyle*,&*ParaUIObject::SetAnimationStyle*)
- (“receivedrag”,&ParaUIObject::GetReceiveDrag,&*ParaUIObject::SetReceiveDrag*)
- (“wordbreak”,&ParaUIObject::GetWordbreak,&ParaUIObject::SetWordbreak)
- (“itemheight”,&ParaUIObject::GetItemHeight,&ParaUIObject::SetItemHeight)
- (“multiselect”,&ParaUIObject::GetMultipleSelect,&ParaUIObject::SetMultipleSelect)
- (“tooltip”,&ParaUIObject::GetToolTip,&ParaUIObject::SetToolTip)
- (“scrollbarwidth”,&ParaUIObject::GetScrollbarWidth,&ParaUIObject::SetScrollbarWidth)
- (“fastrender”,&ParaUIObject::GetFastRender,&ParaUIObject::SetFastRender)
- (“lifetime”,&*ParaUIObject::GetLifeTime*,&*ParaUIObject::SetLifeTime*)
- (“zdepth”,&*ParaUIObject::GetZDepth*,&*ParaUIObject::SetZDepth*)
- (“value”,&*ParaUIObject::GetValue*,&ParaUIObject::SetValue)
- (“fixedthumb”,&ParaUIObject::GetFixedThumb,&ParaUIObject::SetFixedThumb)
- (“thumbsize”,&ParaUIObject::GetThumbSize,&ParaUIObject::SetThumbSize)
- (“tooltip”,&ParaUIObject::GetToolTip,&ParaUIObject::SetToolTip)
- (“canvasindex”,&ParaUIObject::GetCanvasIndex,&ParaUIObject::SetCanvasIndex)
- (“zorder”,&*ParaUIObject::GetZOrder*,&*ParaUIObject::SetZOrder*)
- (“x”,&ParaUIObject::GetX,&ParaUIObject::SetX)

- (“y”,&ParaUIObject::GetY,&ParaUIObject::SetY)
- (“depth”,&ParaUIObject::GetDepth,&ParaUIObject::SetDepth)
- (“width”,&ParaUIObject::Width,&ParaUIObject::SetWidth)
- (“height”,&ParaUIObject::Height,&ParaUIObject::SetHeight)
- (“rotation”,&*ParaUIObject::GetRotation*,&*ParaUIObject::SetRotation*)
- (“scalingx”,&ParaUIObject::GetScalingX,&ParaUIObject::SetScalingX)
- (“scalingy”,&ParaUIObject::GetScalingY,&ParaUIObject::SetScalingY)
- (“translationx”,&ParaUIObject::GetTranslationX,&ParaUIObject::SetTranslationX)
- (“translationy”,&ParaUIObject::GetTranslationY,&ParaUIObject::SetTranslationY)
- (“colormask”,&ParaUIObject::GetColorMask,&*ParaUIObject::SetColorMask*)
- (“spacing”,&*ParaUIObject::GetSpacing*,&*ParaUIObject::SetSpacing*)
- (“popup”,&ParaUIObject::GetPopUp,&ParaUIObject::SetPopUp)
- (“onframemove”,&ParaUIObject::GetOnFrameMove,&ParaUIObject::OnFrameMove)
- (“onclick”,&ParaUIObject::GetOnClick,&*ParaUIObject::OnClick*)
- (“onchange”,&ParaUIObject::GetOnChange,&*ParaUIObject::OnChange*)
- (“onkeydown”,&ParaUIObject::GetOnKeyDown,&*ParaUIObject::OnKeyDown*)
- (“onkeyup”,&ParaUIObject::GetOnKeyUp,&ParaUIObject::OnKeyUp)
- (“ondoubleclick”,&ParaUIObject::GetOnDoubleClick,&*ParaUIObject::OnDoubleClick*)
- (“ondragbegin”,&ParaUIObject::GetOnDragBegin,&ParaUIObject::OnDragBegin)
- (“ondragend”,&ParaUIObject::GetOnDragEnd,&ParaUIObject::OnDragEnd)
- (“ondragmove”,&ParaUIObject::GetOnDragOver,&ParaUIObject::OnDragOver)
- (“onmousedown”,&ParaUIObject::GetOnMouseDown,&ParaUIObject::OnMouseDown)
- (“onmouseup”,&ParaUIObject::GetOnMouseUp,&ParaUIObject::OnMouseUp)
- (“onmousemove”,&ParaUIObject::GetOnMouseMove,&ParaUIObject::OnMouseMove)
- (“onmousewheel”,&ParaUIObject::GetOnMouseWheel,&ParaUIObject::OnMouseWheel)
- (“onmousehover”,&ParaUIObject::GetOnMouseHover,&ParaUIObject::OnMouseHover)
- (“onmouseenter”,&ParaUIObject::GetOnMouseEnter,&ParaUIObject::OnMouseEnter)
- (“onmouseleave”,&ParaUIObject::GetOnMouseLeave,&ParaUIObject::OnMouseLeave)
- (“onselect”,&ParaUIObject::GetOnSelect,&ParaUIObject::OnSelect)
- (“onmodify”,&ParaUIObject::GetOnModify,&ParaUIObject::OnModify)
- (“ondestroy”,&ParaUIObject::GetOnDestroy,&*ParaUIObject::OnDestroy*)
- (“onsize”,&ParaUIObject::GetOnSize,&*ParaUIObject::OnSize*)

Class Properties

Public Functions

bool **IsValid**() **const**
check if the object is valid

Return : true is the object is a valid GUI object.

void **AddChild**(*ParaUIObject pChild*)
Attach a child GUI object to this object This function is only for container objects;

Parameters

- `pChild`: the child object to be attached

ParaAttributeObject **GetAttributeObject**()
get the attribute object associated with an object.

void **GetAttributeObject_**(*ParaAttributeObject &output*)
for API exportation

void **AddTextItem**(**const** char **text*)
Add a text item for listbox

Parameters

- `text`: the text string to be added.

void **AttachToRoot**()
Attach this object to the root of the screen GUI.

string **GetName**() **const**
Get the name of this object

ParaUIObject **GetChild**(**const** char **name*)
get child by name

ParaUIObject **GetChildAt**(int *index*)
get child by index.Index start from 0. Use *GetChildCount()* for total child count.

int **GetChildCount**()
get the total number of child UI objects.

int **GetID**() **const**
get id of this object. please note that this is a child id, not a globally unique id. the id is only available when this object is attached to a parent. And the ID will change if this object changes its parent. In all other circumstances, the id uniquely identify this object in its parent. One can call *GetChildByID()* from its parent control to get this object. *Note*: ID is assigned by its parent when this control is attached to a parent control (or parent changes) it ensures that ChildID is unique among all sibling children of the parent control during the lifetime of the parent.

void **SetID**(int *nID*)
this function is used internally. never call this unless you known why.

ParaScripting::ParaUIObject **GetChildByID**(int *nChildID*)
get a child node by its id

Return : return the child object found. it may return invalid object if not found.

Parameters

- `nChildID`: child ID usually obtained by *GetID()* method.

ParaScripting::*ParaUIObject* **GetChildByName**(**const** char **name*)

get the first child node whose name is name. Since a name may not be unique among its sibling children. One is advised to use *GetChildByID()* instead.

Return : return the child object found. it may return invalid object if not found.

Parameters

- `name`: child name usually obtained by *GetName()* method.

void **SetEnabled**(bool *bEnabled*)

Set if a control is enabled

Parameters

- `bEnabled`: the new value

void **SetUseTextShadow**(bool *bUseTextShadow*)

Set/Get whether the use text shadow

void **SetTextScale**(float *fScale*)

set the text scale the text scale, default to 1.f. if we have text scale between 1.1-1.5 and shadow to true, the text will have an alpha border. This is great for rendering text in 3d scene with a boarder using just normal system font.

float **GetTextScale**()

get the text scale the text scale, default to 1.f. if we have text scale between 1.1-1.5 and shadow to true, the text will have an alpha border. This is great for rendering text in 3d scene with a boarder using just normal system font.

string **GetColor**() **const**

Get the UI control color for the current state.

void **SetColor**(**const** char **strColor*)

set the UI control color for the current state.

Parameters

- `strColor:"255`: 0 0" or "255 0 0 128"

void **SetZDepth**(float *fDepth*)

set z depth in the range [0,1], where 0 is closest to screen. if this is -1(default), zdepth is automatically determined. Otherwise it will force the z depth.

float **GetZDepth**()

get z depth

void **SetAutoSize**(bool *bAutosize*)

Set if a control is auto resize

Parameters

- `bAutosize`: the new value

void **GetTextLineSize**(int **width*, int **height*)

get the text line size in pixels, supposing the current font and text will be rendered in a single line.

void **SetVisible**(bool *bVisible*)

Set if a control is visible

Parameters

- `bVisible`: the new value

void **`SetReceiveDrag`**(bool *bReceiveDrag*)
Get the resource of this control

luabind::object **`GetField`**(**const** char **sFieldname*, **const** object &*output*)
get field by name.

void **`SetField`**(**const** char **sFieldname*, **const** object &*input*)
set field by name

Parameters

- `sFieldname`: field name
- `input`: input value. if field type is vectorN, input is a table with N items.

void **`CallField`**(**const** char **sFieldname*)
call field by name. This function is only valid when The field type is void. It simply calls the function associated with the field name.

void **`SetDefault`**(bool *bDefaultButton*)
set as the default button of the container. please note each container can only have one default button. so as one set a default button, it will automatically unset old default button in the parent container. when enter key is pressed in an edit box, the default button will be clicked.

void **`SetLifeTime`**(int *nSeconds*)
Set a count down timer in frames. when it is zero, the object will be automatically deleted If LifeTime is negative, the object will be permanent.

Parameters

- `nSeconds`: how many seconds the window left to be alive.

int **`GetLifeTime`**() **const**
Get a count down timer in frames. when it is zero, the object will be automatically deleted If LifeTime is negative, the object will be permanent.

Return : return how many seconds the window left to be alive.

string **`GetText`**() **const**
Get text of a control;

void **`SetText3`**(**const** char **strText*, **const** char **strColor*, **const** char **strFontAssetName*)
Show text in the window. The text will fill the entire window. If one want to position text in a window, it must create another window to contain the text, then use the window to position the text.

Parameters

- `strText`: text to display (must be in utf8)
- `strColor`: color of the object. such as "255 255 255".
- `strFontAssetName`: the font asset name, it can be "", in which case the default font will be used.

void **`SetTextAutoTranslate`**(**const** char **strText*, **const** char **sConverter*)
basically this is the same as the SetText method, except that it will automatically translate the text from one language to another. please note that this method does not solve all localization problems, but it allows for quick machine translation .

Parameters

- `strText`: utf8 encoded text
- `sConverter`: it should be one of the supported converter name. a most common usage is "c_S2T" where simplified Chinese character is translated to traditional Chinese. The asset file "locale/c_simpl_to_trad.txt" is used.

void **SetPasswordChar** (string *PasswordChar*)

this function is only supported in the "editbox" control, not the "text" or "IMEEditbox" The PasswordChar property specifies the character displayed in the edit box. For example, if you want asterisks displayed in the password box, specify * for the PasswordChar property in the Properties window. Then, regardless of what character a user types in the text box, an asterisk is displayed.

Remark : Security Note: Using the PasswordChar property on a text box can help ensure that other people will not be able to determine a user's password if they observe the user entering it. This security measure does not cover any sort of storage or transmission of the password that can occur due to your application logic. Because the text entered is not encrypted in any way, you should treat it as you would any other confidential data. Even though it does not appear as such, the password is still being treated as a plain-text string (unless you have implemented some additional security measure).

Parameters

- `PasswordChar`: such as '*'

string **GetPasswordChar** () **const**

get password char

void **SetTopLevel** (bool *value*)

Set a container to be a top-level control or set a top-level control back to normal container.

void **SetBGImage1** (**const** object &*szBackground*)

set the background image of the control.

Note : GUI also supports texture bound with dynamic width and height. such as "Texture/anyfile.dds;0 0 -1 -1", where -1 is replaced with the dynamic width or height when needed. This is specially useful for specifying width and height of a HTTP remote texture where the dimension is not known immediately. things in [] are optional input;if not specified, the rectangle contains the whole picture;

Parameters

- `szBackground`: the texture asset file name with the rectangle information the format of szBackground is filename[; left top width height][:left top toright to bottom]. if it is "" or width or height is zero, the texture is not drawn or is fully transparent. e.g. "texture/whitedot.png", "texture/whitedot.png;0 0 64 64", "texture/whitedot.png:10 10 10 10", "texture/whitedot.png;0 0 64 64;10 10 10 10" [:left top toright to bottom] is used to specify the inner rect of the center tile in a nine texture element UI object

void **SetBGImageStr** (**const** char **szBackground*)

szBackground can be filename[; left top width height][:left top toright to bottom], such as "texture/whitedot.png", "texture/whitedot.png;0 0 64 64", "texture/whitedot.png:10 10 10 10", "texture/whitedot.png;0 0 64 64;10 10 10 10" [:left top toright to bottom] is used to specify the inner rect of the center tile in a nine texture element UI object

void **SetNineElementBG** (**const** char **TextFilename*, int *left*, int *top*, int *toRight*, int *toBottom*)

drawing the background using nine element.

Note : Right now, this is only valid on container, but we will support it for all other controll.

Parameters

- `TextFilename`: the background texture to use. it will be divided into 3x3 tiles when drawing the control
- `lefttoptoRighttoBottom`: the inner tile position relative to the dimension of TextFilename texture. If all are 0, it will be set back to one element background

int **`GetFirstVisibleCharIndex`**()
this method is only valid if the control is edit box or imeedit box if the text is longer than the edit box, the returned value is the index of the first visible character in the edit box Normally, this is 0 if the text can be fully contained in the edit box.

int **`GetCaretPosition`**()
return Caret position in characters

void **`SetCaretPosition`**(int *nCharacterPos*)
Set Caret position in characters

Parameters

- `nCharacterPos`: in characters

int **`GetTextSize`**()
return the text size in Unicode character count.

void **`GetPriorWordPos`**(int *nCP*, int *PriorIn*, int &*Prior*)
get the prior word position

void **`GetNextWordPos`**(int *nCP*, int *NextIn*, int &*Next*)
get the next word position

void **`CPtoXY`**(int *nCP*, bool *bTrail*, int *XIn*, int *YIn*, int &*X*, int &*Y*)
Character position to X,Y in pixel

void **`XYtoCP`**(int *nX*, int *nY*, int *CPIn*, int *nTrailIn*, int &*CP*, int &*nTrail*)
X,Y in pixel to character position

void **`SetPageSize`**(int *pagesize*)
Get the page size of the scroll bar Page size is the size of how many items a page contains. The control will scroll a page size if we click on the empty space of the track of the scroll bar

void **`SetStep`**(int *nDelta*)
how many pixels to scroll each step

int **`GetStep`**()
how many pixels to scroll each step

void **`SetTrackRange`**(int *nBegin*, int *nEnd*)
set the page size of a container or a scroll bar

Parameters

- `nBegin`: the start value of the range
- `nEnd`: the end value of the range

int **`GetAnimationStyle`**() **const**
get the animation style of this object. Different GUI object may have different style definitions. for "button" object: 1 is gradually enlarge 5% when getting focus.2 is 10%, 3 is 15%, 4 is 20%, 5 is 25%

Return : 0 always means no animation. 1 is gradually enlarge 5% when getting focus.2 is 10%, 3 is 15%, 4 is 20%, 5 is 25% 11-15 means the same as 1-5, except that the normal state alpha is the same as the highlighted state. 21-25 means the same as 11-15, except that the button animation will not stop in the highlighted state. 31-39 is clock-wise rotation, the larger the faster 41-49 is counter-clock-wise rotation, the larger the faster

void **`SetAnimationStyle`**(int *nStyle*)
set the animation style of this object. Different GUI object may have different style definitions. for "button" object: 1 is gradually enlarge 5% when getting focus.2 is 10%, 3 is 15%, 4 is 20%, 5 is 25%

Parameters

- `nStyle`: 0 always means no animation. 1 is gradually enlarge 5% when getting focus.2 is 10%, 3 is 15%, 4 is 20%, 5 is 25% 11-15 means the same as 1-5, except that the normal state alpha is the same as the highlighted state. 21-25 means the same as 11-15, except that the button animation will not stop in the highlighted state. 31-39 is clock-wise rotation, the larger the faster 41-49 is counter-clock-wise rotation, the larger the faster

void **`SetScrollable`**(bool *bScrollable*)
Set if a container is scrollable

Parameters

- `bScrollable`: the new scrollable value

void **`GetAbsPosition`**(float &*x*, float &*y*, float &*width*, float &*height*, float &*z*) **const**
get the absolute position of the control in screen coordinate e.g. local x,y = obj:*GetAbsPosition()*;

Parameters

- `&x`: screen x
- `&y`: screen y
- `&width`: width
- `&height`: height
- `&z`: depth, usually not used

void **`SetPosition`**(**const** char **pos*)
set rect position of the control.

Parameters

- `pos`: in format "x y width height"

string **`GetPosition`**() **const**
get rect position of the control

void **`Reposition`**(**const** char **alignment*, int *left*, int *top*, int *width*, int *height*)
reposition the control using the same parameter definition used when control is created. see *ParaUI::CreateUIObject()* for parameter definition.

int **`GetValue`**() **const**
Gets and Sets the value of a control Controls that have value are CGUIScrollBar, CGUISlider

void **`OnClick`**(**const** object &*strScriptName*)
When the user clicked on the window, the script file or function will be activated. The following parameters are passed as global variables. id = int; the id of the triggering GUI window mouse_button = ["left"|"right"|"middle"]; mouse_x = number; mouse_y = number;

Parameters

- `strScriptName`: the NPL script file to activate.

void **`OnSize`**(**const** object &*strScriptName*)
called when the size of the window is changed. id = int; the id of the triggering GUI window

void **`OnDestroy`**(**const** object &*strScriptName*)
called when the window is destroyed. Please note that GetUIObject() will return invalid object.

void **`OnActivate`**(**const** object &*strScriptName*)
The framework calls this member function when a window is being activated or deactivated. id = int; the id of the triggering GUI window param1: Specifies whether the window is being activated or deactivated. It can be one of the following values:

- 0 The window is being deactivated.
- 1 The window is being activated by a mouse click.
- 2 The window is being activated through some method other than a mouse click.

void **`OnDoubleClick`**(**const** object &*strScriptName*)
The following parameters are passed as global variables. id = int; the id of the triggering GUI window mouse_button = ["left"|"right"|"middle"]; mouse_x = number; mouse_y = number;

Parameters

- `strScriptName`:

void **`OnKeyDown`**(**const** object &*strScriptName*)
When the user pressed a key while the mouse cursor is inside the window, a script file or function will be activated. The following parameters are passed as global variables. id = int; the id of the triggering GUI window keystring = string;it is filled with the key character. e.g. "\r", "a", etc. It may contain multiple characters, if the user is typing fast or Multi-byte such as Chinese Character is entered.

Parameters

- `strScriptName`: the NPL script file to activate.

void **`SetCursor`**(**const** object &*szCursorFile*)
Set/Get cursor file when mouse is over it. If empty, the parent cursor file is used.

void **`SetCursorEx`**(**const** char **szCursorFile*, int *nHotSpotX*, int *nHotSpotY*)
Set cursor by file and hot spot.

void **`OnChange`**(**const** object &*strScriptName*)
usually the editbox will call this handle, "virtual_key" contains the last key stroke. Usually application may check if it is an enter key for special actions.

void **`SetRotation`**(float *fRot*)
set the rotation of the control around the center of the UI plus rotation origin offset. it only affects the drawing rect but not the mouse sensor rect. it is usually used for visual effect, so there is no need to update or calculate client rect.

float **`GetRotation`**() **const**
Get the rotation of the control around the center of the UI plus rotation origin offset. it only affects the drawing rect but not the mouse sensor rect. it is usually used for visual effect, so there is no need to update or calculate client rect.

void **SetRotOriginOffset** (float *x*, float *y*)
rotation origin offset from the center of the UI object.

void **GetRotOriginOffset** (float **x*, float **y*) **const**
rotation origin offset from the center of the UI object.

void **SetScaling** (float *x*, float *y*)
set the scaling of the control around the center of the UI plus rotation origin offset. it only affects the drawing rect but not the mouse sensor rect. it is usually used for visual effect, so there is no need to update or calculate client rect.

void **GetScaling** (float **x*, float **y*) **const**
Get the scaling of the control around the center of the UI plus rotation origin offset. it only affects the drawing rect but not the mouse sensor rect. it is usually used for visual effect, so there is no need to update or calculate client rect.

void **SetTranslation** (float *x*, float *y*)
set the translation of the control around the center of the UI plus rotation origin offset. it only affects the drawing rect but not the mouse sensor rect. it is usually used for visual effect, so there is no need to update or calculate client rect.

void **GetTranslation** (float **x*, float **y*) **const**
Get the translation of the control around the center of the UI plus rotation origin offset. it only affects the drawing rect but not the mouse sensor rect. it is usually used for visual effect, so there is no need to update or calculate client rect.

void **SetColorMask** (**const** char **strColor*)
color mask, no matter what state the control is in. “r g b a”, such as “255 255 255 255”

void **ApplyAnim** ()
automatically animate the child nodes according to this object’s rotation, scaling, translation and color mask values.

void **SetSpacing** (int *nSpacing*)
Spacing between the text and the edge of border

int **GetSpacing** () **const**
Spacing between the text and the edge of border

void **AttachTo3D** (**const** *ParaObject* *obj*)
Attach the control to the 3D object. This function will internally call *AttachToRoot()* if it has not been called previously One can call this function multiple times move the attached GUI object from one 3D object to another.

Remark : The visible property of this GUI object is controlled internally as below:

- if the 3d object does not exist, the GUI object is invisible.
- if the 3d object is not visible in the current camera frustum, the GUI object is invisible.
- if the 3d object is visible in the current camera frustum, the GUI object is visible.

Parameters

- `obj`: this must be a global object

void **AttachTo3D_** (**const** char **s3DObjectName*)
save as AttachTo3D, except that the 3D object’s name is specified.

Remark : The visible property of this GUI object is controlled internally as below:

- if the 3d object does not exist, the GUI object is invisible.
- if the 3d object is not visible in the current camera frustum, the GUI object is invisible.
- if the 3d object is visible in the current camera frustum, the GUI object is visible.

Parameters

- `s3DObjectName`: this must be the name of a global object.

ParaUIFont **`GetFont`**(**const** char **name*)
get font by name. The prefered way is to get by index.

Parameters

- `name`: this is usually "text".

ParaUIFont **`GetFont_`**(int *nIndex*)
get font by index

void **`DoAutoSize`**()
try to automatically adjust the size of this object to contain all its content.

ParaUITexture **`GetTexture`**(**const** object &*name*)
Get the texture object associated with the current control A control may be associated with multiple textures.

Return

Parameters

- `name`: different controls have set of textures. it can be one of the following predefined string.
  - "background": the background texture, this is the most commonly used texture and is available in all controls.
  - "track": the track texture in a scroll bar. it is available in "scrollbar" control.
  - "up_left": the up arrow texture in scroll bar. it is available in "scrollbar" control.
  - "down_left": the down arrow texture in scroll bar. it is available in "scrollbar" control.
  - "thumb": the thumb button texture in the scroll bar. it is available in "scrollbar" control.

void **`BringToFront`**()
bring to front among the same z order

void **`BringToBack`**()
bring to back among the same z order

int **`GetZOrder`**() **const**
z order affect the order of rendering. The parent control sort and render child control from small z value to larger z value. default value is 0.

void **`SetZOrder`**(int *nOrder*)
z order affect the order of rendering. The parent control sort and render child control from small z value to larger z value. default value is 0. if this control has a parent it will cause the parent to sort all children again.

void **`SetActiveLayer`**(**const** char **layer*)
Set the active layer for setting resource;

Parameters

- `layer`: name of the layer We have three layers "artwork", "background" and "overlay" "artwork" layer is the primary layer. By default, we render a control using this layer "overlay" layer is a layer on top of the "artwork" layer. "background" layer is on the bottom of the "artwork" layer. All three layers are the same. If not implicitly set, the default active layer is the "artwork" layer. But when a user wants to set something, remember to call this function to ensure that the active layer is correct.

bool **HasLayer**(**const** char **layer*)

return whether a given layer exist in the resource. One needs to call *SetActiveLayer()* in order to create a layer.

Parameters

- `layer`: it can be one of the three layers "artwork", "background" and "overlay". Typically, all controls contains the "artwork" layer.

void **SetCurrentState**(**const** char **statename*)

Set the current state for setting resource:

Parameters

- `statename`: name of the state We have four states "normal", "highlight", "pressed", "disabled" "normal" state is the default state. "highlight" state is the highlighted state. Some controls will change their state to "highlight" automatically when mouse enters them. "pressed" state is when a button or something else is pressed. "disabled" state is when a control is disabled. If not implicitly set, the default state is the "normal" state. But when a user wants to set something, remember to call this function to ensure that the current state is correct.

void **CloneState**(**const** char **statename*)

Clones a state to the current state

Parameters

- `statename`: name of the state We have four states "normal", "highlight", "pressed", "disabled" "normal" state is the default state. "highlight" state is the highlighted state. Some controls will change their state to "highlight" automatically when mouse enters them. "pressed" state is when a button or something else is pressed. "disabled" state is when a control is disabled.

void **InvalidateRect**()

the window's client(child) area that must be recalculated and redrawn this function is obsoleted, call *UpdateRect()* instead

void **UpdateRect**()

recalculated the window's client rect

class ParaUITexture

#include <ParaScriptingGraphics.h> a GUI texture object

Class Properties

- ("transparency",&ParaUITexture::GetTransparency,&ParaUITexture::SetTransparency)
- ("color",&ParaUITexture::GetColor,&ParaUITexture::SetColor)
- ("texture",&ParaUITexture::GetTexture,&ParaUITexture::SetTexture)
- ("rect",&ParaUITexture::GetTextureRect,&ParaUITexture::SetTextureRect),

Public Functions

bool **IsValid**()
check if the object is valid

class **ParaWorld**
#include <ParaScriptingWorld.h> world creation functions.

Public Static Functions

string **NewWorld**(**const** char **sWorldName*, **const** char **sBaseWorldName*)
created a new empty world based on a given world. This is like class inheritance in C++ another world could derive from a given world, overriding certain terrain tile contents as it evolves. One can immediately create the new world which is identical to the base world. Every changes made to the new world will be saved in the new world folder and will not affect the base world. in reality, we just copy the base world's world config file to the new world using the new world's name. e.g local sConfigFileName = *ParaWorld.NewWorld*("__TmpWorld", "sample/worldconfig.txt"); if(sConfigFileName ~= "") then*ParaScene.CreateWorld*("", 32000, sConfigFileName); end NOTE: if the sWorldName is the same as sBaseWorldName, the sBaseWorldName itself will be overridden

Return : return the world config file. if failed, return ""

Parameters

- `sWorldName`: world name, a directory with the same name will be created containing the world config file.
- `sBaseWorldName`: any valid world config file.

const char ***NewWorld_**(**const** char **sWorldName*, **const** char **sBaseWorldName*)
solely used for exporting

void **DeleteWorld**(**const** char **sWorldName*)
delete a given world.

string **NewEmptyWorld**(**const** char **sWorldName* = NULL, float *fTileSize* = 533.3333f, int *nTileDimension* = 64)
Create an empty world, with flat land.

Parameters

- `sWorldName`: world name, if NULL, it defaults to "_emptyworld"
- `fTileSize`: terrain tile size in meters
- `nTileDimension`: dimension of the tile matrix. default is 64, which is has 64*64 tiles

const char ***NewEmptyWorld_**(**const** char **sWorldName* = NULL, float *fTileSize* = 533.3333f, int *nTileDimension* = 64)
solely used for exporting

void **SetEnablePathEncoding**(bool *bEnable*)
set whether we will encode world related files. default to true. By enabling path encoding, world related files like "worlddir/worldfile.txt" will be saved as "%WORLD%/worldfile.txt", thus even the entire world directory changes, the world files can still be found using path variables. Path encoding needs to be disabled when you are creating a template world.

bool **`GetEnablePathEncoding`**()
: get whether we will encode world related files. default to true. By enabling path encoding, world related files like "worlddir/worldfile.txt" will be saved as "%WORLD%/worldfile.txt", thus even the entire world directory changes, the world files can still be found using path variables. Path encoding needs to be disabled when you are creating a template world.

ParaDataProvider **`GetNpcDB`**()
: get the current NPC data provider.

void **`GetNpcDB_`**(ParaDataProvider **out*)
: solely used for exporting

void **`SetNpcDB`**(**const** char **sConnectionstring*)
: set the global NPC data base to a new database file.

Parameters

- `sConnectionstring`: : currently it is the file path of the database file.

ParaDataProvider **`GetAttributeProvider`**()
: get the current attribute data provider.

void **`GetAttributeProvider_`**(ParaDataProvider **out*)
: solely used for exporting

void **`SetAttributeProvider`**(**const** char **sConnectionstring*)
: set the current attribute data base to a new database file.

Parameters

- `sConnectionstring`: : currently it is the file path of the database file.

void **`SetWorldDB`**(**const** char **sConnectionstring*)
: set world database. it sets attribute provider, NPC database, etc to the same database file.

Parameters

- `sConnectionstring`: : currently it is the file path of the database file.

string **`GetStringbyID`**(int *ID*)
: Get string from ID

Return string in the current game language

Parameters

- `ID`: ID in kids db's string table.

int **`InsertString`**(**const** char **strEN*, **const** char **strCN*)
: Insert the new string table entry to StringTable_DB

Return ID of the inserted string

Parameters

- `str`: Entry in the current game language

void **`SetServerState`**(int *nState*)
: set the current server state. default value is 0. enum CurrentState { STATE_STAND_ALONE = 0, STATE_SERVER, STATE_CLIENT, };

bool **`SendTerrainUpdate`**(**const** char **sDestination*, float *center_x*, float *center_y*, float *center_z*, float *fRadius*, float *fResolution*)

Call this function to send a copy of the local terrain to a destination world at a given resolution.

Return

Parameters

- `sDestination`: a destination namespace, it could be "all@server" or "@server", etc. This will be used to fill the destination address of the packet to be sent.
- `center_x`: center of the terrain region to send
- `center_y`: center of the terrain region to send
- `center_z`: center of the terrain region to send
- `fRadius`: radius of the terrain region to send
- `fResolution`: if this is 1.f, the local world terrain grid resolution is used.

string **`GetWorldName`**()

get current world name

string **`GetWorldDirectory`**()

get world root path. suppose the given world name is "sample". The generated file name is "sample/"

void **`SetScriptSandBox`**(**const** object *&SandboxNeuronFile*)

Paraworld is told to execute in the given sandbox. (1) *ParaWorld* namespace supports a sandbox mode, which can be turned on and off on demand. Once turned on, all scripts from the current game world will be executed in a separate and newly created script runtime environment. (2) Sandbox mode is an isolated mode that does not have any link with the glia file environment. (3) The world scripts protected by the sandbox runtime environment includes: terrain tile onload script, biped event handler scripts, such as character onload, onclick events. (4) The sandbox environment includes the following functions that could be used: *ParaScene*, *ParaUI* namespace functions. It also explicitly disabled the following functions: a) Dofile() b) Io, *ParaIO*, Exec c) Require(),NPL.load, NPL.activate, NPL.download: cut off any way to manually load a file. It adds d) Log e) Download some file to replace local file. f) Changing the Enter sand box function or almost any function to some fake function. (5) The following attack methods should be prevented by the sandbox environment a) Execute or load any external application b) Write to any file, including log file c) Compromise data or functions in the glia file environment. Such as, changing and hooking the string method d) Compromise the sandbox itself and then affect in the next sandbox entity. (6) glia file environment does not have a sandbox mode. Because I found that any global sandbox mode implementation has a way to hack in, and I give up any measure of protecting the glia file environment. Sandbox protection for the world file is enough because that is the only source file that may not be provided by ParaEngine. In order to run any other code not provided by the ParaEngine, the user should have been well informed of the danger. But so far, there is no need to have a world that should inform the user. Because most world functions are supported in the world sandbox.

Parameters

- `sNeuronFileName`: script file name. Use NPL.CreateNeuronFile() to create a sandbox neuron file if you do not want to use a sandbox, please specify NULL or "".

const char ***`GetScriptSandBox`**()

please see *SetScriptSandBox()*

Return : it may return NULL if the sandbox does not exist.

class **ParaXML**

#include <ParaScriptingIO.h> *ParaXML* class

Public Static Functions

int **LuaXML_ParseString**(lua_State **L*)

only used for documentation generation. Spec by example lz = *ParaXML.LuaXML_ParseString*[[<paragraph justify="centered">first child**bold**second child</paragraph>]] lz = {name="paragraph", attr={justify="centered"}, [1] = "first child", [2] = {name="b", "bold", n=1} [3] = "second child", n=3 }

Return return a table containing the lua table of xml Specifications: A tree is a Lua table representation of an element and its contents. The table must have a name key, giving the element name. The tree may have a attr key, which gives a table of all of the attributes of the element. Only string keys are relevant. If the element is not empty, each child node is contained in tree[1], tree[2], etc. Child nodes may be either strings, denoting character data content, or other trees.

Parameters

- `filename`: string

void **SetCondenseWhiteSpace**(bool *condense*)

The world does not agree on whether white space should be kept or not. In order to make everyone happy, these global, static functions are provided to set whether or not the parser will condense all white space into a single space or not. The default is to condense. Note changing this value is not thread safe.

bool **IsWhiteSpaceCondensed**()

Return the current white space setting.

class **ParaZipWriter**

#include <ParaScriptingIO.h> *ParaZipWriter* class: creating zip files

e.g. (1) Traditional use, creating a zipfile from existing files local writer = *ParaIO.CreateZip*("c:\\simple1.zip",""); writer:ZipAdd("znsimple.bmp", "c:\\simple.bmp"); writer:ZipAdd("znsimple.txt", "c:\\simple.txt"); writer:*close()*;

Public Functions

bool **IsValid**()

whether it is valid

DWORD **ZipAdd**(**const** char **dstzn*, **const** char **fn*)

add a zip file to the zip. file call this for each file to be added to the zip.

Return : 0 if succeed.

DWORD **ZipAddFolder**(**const** char **dstzn*)

add a zip folder to the zip file. call this for each folder to be added to the zip.

Return : 0 if succeed.

DWORD **AddDirectory**(**const** char **dstzn*, **const** char **filepattern*, int *nSubLevel* = 0)

add everything in side a directory to the zip. e.g. AddDirectory("myworld/", "worlds/myworld/*.*", 10);

Parameters

- `dstzn`: all files in fn will be appended with this string to be saved in the zip file.
- `filepattern`: file patterns, which can include wild characters in the file portion.
- `nSubLevel`: sub directory levels. 0 means only files at parent directory.

DWORD **`close`**`()`

call this when you have finished adding files and folders to the zip file. Note: you can't add any more after calling this.

template <typename *_Ty*>
class `StackObjectPtr`

#include <ParaScriptingGlobal.h> this is for luabinding a pointer object on lua stack, instead of invoking new operator each time an object is created. Note: this class is not used, since we now uses a more universal template which treats all objects in *ParaScripting* namespace as stack object.

Index