antara-gaming-sdk

Release 0.0.2

Roman Sztergbaum Tolg Ay

Oct 06, 2019

CONTENTS

1 Introduction 3

2 Indices and tables 21

Index 23

CHAPTER

ONE

INTRODUCTION

Welcome to the documentation of the antara-gaming-sdk. `antara-gaming-sdk` is an SDK programmed in **C++ 17** that aims to be very fast in runtime by using the maximum of features at the compilation while remaining extensible to the runtime. It is based on an architecture of *modules* that can be used separately or together.

Below a list of topics covered by the documentation:

1.1 Antara Gaming API documentation

1.1.1 antara::gaming::config

template<typename **TConfig**>
TConfig `antara::gaming::config::`**`load_configuration`**(std::filesystem::path *&&config_path*, std::string *filename*)

This function allows us to load a configuration through a `path` and `filename`. There are three different behaviors in this function:

- if the parameter path does not exist the function will attempt to create the directories of the given `path`.
- if the configuration does not exist a default one will be **created**.
- if the `path` and the `name` of the file exists, the contents of the configuration will be **loaded**.

Example:

```
auto cfg = config::load_configuration<my_game::config>(std::filesystem::current_
↪path() / "assets/config", "my_game.config.json");
```

Return a loaded/created configuration.

Template Parameters

- `TConfig`: the type of template you want to load

Parameters

- `config_path`: the path to the configuration you want to load
- `filename`: the name of the configuration you want to load.

1.1.2 antara::gaming::core

constexpr const char *`antara::gaming::version()`

Function that allows us to find the current version of the SDK.

Example:

```
#include <iostream>
#include <antara/gaming/core/version.hpp>

void print_version() {
    std::cout << antara::gaming::version() << std::endl;
}
```

Return the current version of the SDK as a `const char *`

Note: The result of this function can be deduced at compile-time.

1.1.3 antara::gaming::ecs

antara::gaming::ecs::system_manager

class system_manager

This class allows the manipulation of systems, the addition, deletion, update of systems, deactivation of a system, etc.

Public Types

using system_ptr = std::unique_ptr<base_system>

sugar name for a pointer to base_system

using system_array = std::vector<*system_ptr*>

sugar name for an array of system_ptr

using system_registry = std::array<*system_array*, system_type::size>

sugar name for a multidimensional array of system_array (pre_update, logic_update, post_update)

Public Functions

system_manager(entt::registry &*registry*, bool *subscribe_to_internal_events* = true)

Constructors.

Parameters

- `registry`: The entity_registry is provided to the system when it is created.
- `subscribe_to_internal_events`: Choose whether to subscribe to default *system_manager* events

Note: Principal Constructor.

Example:

```
#include <entt/entity/registry.hpp>
#include <entt/dispatcher/dispatcher.hpp>
#include <antara/gaming/ecs/system.manager.hpp>

int main()
{
    entt::registry entity_registry;
    entt::dispatcher& dispatcher{registry.set<entt::dispatcher>()};
    antara::gaming::ecs::system_manager mgr{entity_registry};
}
```

void **receive_add_base_system**(**const** ecs::event::add_base_system *&evt*)

Callback.

Parameters

- `evt`: The event that contains the system to add

void **start**()

This function tells the system manager that you start your game.

Note: This function, which indicates the game is spinning, allows actions to be done at each end of the frame like delete systems or add them while we are going to iterate on

Example:

```
#include <entt/entity/registry.hpp>
#include <entt/dispatcher/dispatcher.hpp>
#include <antara/gaming/ecs/system.manager.hpp>

int main()
{
    entt::registry entity_registry;
    entt::dispatcher dispatcher
    antara::gaming::ecs::system_manager system_manager{entity_registry,␣
↪dispatcher};
    system_manager.start();
    return 0;
}
```

std::size_t **update**()

Return number of systems which are successfully updated

Note: This is the function that update your systems. Based on the logic of the different kinds of antara systems, this function takes care of updating your systems in the right order.

Warning: If you have not loaded any system into the system_manager the function returns 0. If you decide to mark a system, it's automatically deleted at the end of the current loop tick through

this function. If you decide to add a system through an *ecs::event::add_base_system event*, it's automatically added at the end of the current loop tick through this function.

Example:

```
#include <entt/entity/registry.hpp>
#include <entt/dispatcher/dispatcher.hpp>
#include <antara/gaming/ecs/system.manager.hpp>

int main()
{
    entt::registry entity_registry;
    entt::dispatcher dispatcher
    antara::gaming::ecs::system_manager system_manager{entity_registry,␣
↪dispatcher};
    system_manager.start();
    // ... added 5 differents systems here
    std::size_t nb_systems_updated = system_manager.update();
    if (nb_systems_updated != 5) {
       // Oh no, i expected 5 systems to be executed in this game loop␣
↪tick
    }
    return 0;
}
```

std::size_t **update_systems**(system_type *system_type_to_update*)

Return number of systems which are successfully updated

Note: This function is called multiple times by update(). It is useful if you want to program your own update function without going through the one provided by us.

See *update*

Parameters

- `system_type_to_update`: kind of systems to update (pre_update, logic_update, post_update)

template<typename **TSystem**>
const *TSystem* &**get_system**() **const**

This function allows you to get a system through a template parameter.

Return A reference to the system obtained.

Template Parameters

- `TSystem`: represents the system to get.

template<typename **TSystem**>
TSystem &**get_system**()

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Example:

```
#include <entt/entity/registry.hpp>
#include <entt/dispatcher/dispatcher.hpp>
#include <antara/gaming/ecs/system.manager.hpp>

int main()
{
    entt::registry entity_registry;
    entt::dispatcher dispatcher
    antara::gaming::ecs::system_manager system_manager{entity_registry,␣
↪dispatcher};
    system_manager.start();
    // ... added 2 differents systems here (render_system, and a log_system)
    auto& render_system = system_manager.get_system<game::render_system>();

    const auto& log_system = system_manager.get_system<game::log_system>();
    return 0;
}
```

template<typename ...**TSystems**>
std::tuple<std::add_lvalue_reference_t<*TSystems*>...> **get_systems**()

This function allow you to get multiple system through multiple templates parameters.

Note: This function recursively calls the get_system function Based on the logic of the different kinds of antara systems, this function takes care of updating your systems in the right order.

Return Tuple of systems obtained.

See *get_system*

Template Parameters

- `TSystems`: represents a list of systems to get

template<typename ...**TSystems**>
std::tuple<std::add_lvalue_reference_t<std::add_const_t<*TSystems*>>...> **get_systems**() **const**

const version overload of get_systems

Note: This function is marked as nodiscard.

Example:

```
#include <entt/entity/registry.hpp>
#include <entt/dispatcher/dispatcher.hpp>
#include <antara/gaming/ecs/system.manager.hpp>

int main()
{
    entt::registry entity_registry;
    entt::dispatcher dispatcher
    antara::gaming::ecs::system_manager system_manager{entity_registry,␣
↪dispatcher};
    // ... added differents systems here
    // Called from a const context
    auto &&[system_foo, system_bar] = system_manager.get_systems<system_foo,␣
↪system_bar>();
```

(continues on next page)

(continued from previous page)

```
    // Called from a non const context
    auto&&[system_foo_nc, system_bar_nc] = system_manager.get_systems<system_
↪foo, system_bar>();

    // Get it as a tuple
    auto tuple_systems = system_manager.get_systems<system_foo, system_bar>();
    return 0;
}
```

See *get_systems*

template<typename **TSystem**>
bool **has_system**() **const**

This function allow you to verify if a system is already registered in the *system_manager*.

Note: This function is marked as nodiscard.

Return true if the system has been loaded, false otherwise

Template Parameters

- `TSystem`: Represents the system that needs to be verified

Example:

```
#include <entt/entity/registry.hpp>
#include <entt/dispatcher/dispatcher.hpp>
#include <antara/gaming/ecs/system.manager.hpp>

int main()
{
    entt::registry entity_registry;
    entt::dispatcher dispatcher
    antara::gaming::ecs::system_manager system_manager{entity_registry,␣
↪dispatcher};

    bool result = system_manager.has_system<my_game::render_system>();
    if (!result) {
        // Oh no, i don't have a rendering system.
    }
    return 0;
}
```

template<typename ...**TSystems**>
bool **has_systems**() **const**

This function allow you to verify if a list of systems is already registered in the *system_manager*.

Note: This function is marked as nodiscard. This function recursively calls the has_system function.

Return true if the list of systems has been loaded, false otherwise

See *has_system*

Template Parameters

- `TSystems`: represents a list of system that needs to be verified

Example:

```
#include <entt/entity/registry.hpp>
#include <entt/dispatcher/dispatcher.hpp>
#include <antara/gaming/ecs/system.manager.hpp>

int main()
{
    entt::registry entity_registry;
    entt::dispatcher dispatcher
    antara::gaming::ecs::system_manager system_manager{entity_registry,␣
↪dispatcher};

    bool result = system_manager.has_systems<my_game::render_system, my_
↪game::input_systems>();
    if (!result) {
        // Oh no, atleast one of the systems is not present
    }
    return 0;
}
```

template<typename **TSystem**>
bool **mark_system**()

Note This function marks a system that will be destroyed at the next tick of the game loop.

Return true if the system has been marked, false otherwise

Template Parameters

- `TSystem`: Represents the system that needs to be marked

template<typename ...**TSystems**>
bool **mark_systems**()

This function recursively calls the mark_system function

Note This function marks a list of systems, marked systems will be destroyed at the next tick of the game loop.

Return true if the list of systems has been marked, false otherwise

See *mark_system*

Template Parameters

- `TSystems`: Represents a list of systems that needs to be marked

template<typename **TSystem**>
bool **enable_system**()

Note This function enable a system

Return true if the system has been enabled, false otherwise

Template Parameters

- `TSystem`: Represents the system that needs to be enabled.

template<typename ...**TSystems**>

bool **enable_systems**()

This function recursively calls the enable_system function

Note This function enable a list of systems

Return true if the list of systems has been enabled, false otherwise

See *enable_system*

Template Parameters

- `TSystems`: Represents a list of systems that needs to be enabled

template<typename **TSystem**>
bool **disable_system**()

Note This function disable a system

Return true if the the system has been disabled, false otherwise

Attention If you deactivate a system, it will not be destroyed but simply ignore during the game loop

Template Parameters

- `TSystem`: Represents the system that needs to be disabled

template<typename ...**TSystems**>
bool **disable_systems**()

This function recursively calls the disable_system function

Note This function disable a list of systems

Return true if the list of systems has been disabled, false otherwise

Template Parameters

- `TSystems`: Represents a list of systems that needs to be disabled

std::size_t **nb_systems**() **const**

Return number of systems

std::size_t **nb_systems**(system_type *sys_type*) **const**

Return number of systems of a specific type.

Parameters

- `sys_type`: represent the type of systems.

template<typename **TSystem**, typename ...**TSystemArgs**>
TSystem &**create_system**(*TSystemArgs*&&... *args*)

Note This function allow you to create a system with the given argument

Note This function is a factory

Return Returns a reference to the created system

Template Parameters

- `TSystem`: represents the type of system to create
- `TSystemArgs`: represents the arguments needed to construct the system to create

template<typename ...**TSystems**, typename ...**TArgs**>

auto **load_systems** (*TArgs*&&... *args*)

Return Tuple of systems loaded

See *create_system*

Template Parameters

- `TSystems`: represents a list of systems to be loaded

Private Types

using clock = std::chrono::steady_clock
Private data members.

Private Functions

base_system &**add_system_** (*system_ptr* &&*system*, system_type *sys_type*)
Private member functions.

1.2 Antara Gaming Tutorials

1.2.1 Tutorial: Getting Started

1.2.2 antara-gaming-sdk

Antara Gaming Software Development Kit

Prerequisites

Below is the list of prerequisites to use the `antara-gaming-sdk` on your machine:

- CMake 3.14 minimum
- **clang-8** minimum (Windows/Linux/Osx)
- **Optional** emscripten latest (Web)
- **Optional** Visual Studio 2019
- **Optional** Clang VS Toolset (installable through visual studio installer)

Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

See deployment for notes on how to deploy the project on a live system.

Build

To build the project please follow the instructions below:

```
mkdir build ## bash or powershell
cd build ## bash or powershell

## Release or Debug are available
cmake -DCMAKE_BUILD_TYPE=Debug -DCMAKE_CXX_COMPILER=your_path_to_your_clang++ ../
↪#Linux / Osx
cmake -DCMAKE_BUILD_TYPE=Debug -G "Visual Studio 16 2019" -A x64 -T "ClangCl" -DCMAKE_
↪CXX_COMPILER="C:/Program Files/LLVM/bin/clang-cl.exe" ../ #Windows

## Build (Debug / Release available)
cmake --build . --config Debug
```

Installing

You do not need to install the gaming sdk, just use the CMake fetch_content command to use the project

Running the tests

Once you have compiled the sdk gaming with the option to enable unit tests.

They are located in the `bin/unit_tests` (Linux/Osx) or `bin/unit_tests/%CMAKE_BUILD_TYPE%` (Windows) folder

Deployment

construction

Built With

- doctest - The fastest feature-rich C++11/14/17/20 single-header testing framework for unit tests and TDD http://bit.ly/doctest-docs
- doom-st - C++ implementation of strong types
- EnTT - Gaming meets modern C++ - a fast and reliable entity-component system (ECS).
- refl-cpp A compile-time reflection library for modern C++
- range-v3 Range library for C++14/17/20, basis for C++20's std::ranges
- expected C++11/14/17 std::expected with functional-style extensions https://tl.tartanllama.xyz
- (optional)SFML Simple and Fast Multimedia Library http://www.sfml-dev.org/

Code of Conduct

Before any contribution please read our CODE OF CONDUCT.

Contributing

Please read CONTRIBUTING.md, contain the process for submitting pull requests to us.

Versioning

We use SemVer for versioning. For the versions available, see the tags on this repository.

Authors

- **Roman Sztergbaum** - *Co-Creator & Lead Dev* - Milerius
- **Tolga Ay** - *Co-Creator* - naezith

See also the list of contributors who participated in this project.

Acknowledgments

- Viktor Kirilov onqtam for the awesome `doctest` framework.
- Michele Caini skypjack for the awesome `EnTT` framework and his help.

Badges

Apps	Badges
Github Actions CI (Windows/Osx/Linux)	
License	This work is licensed under a .
LGTM (Security C++) Alerts	
LGTM (Security C++) Quality	
Issues	
Report CI (Linux/Osx/Windows/Emscripten)	
Coverage (Codecov)	
Docs	
HitCount	
Line Of Code	
Conventional Commit	

1.2.3 Tutorial: Quick And Dirty

If you have not read the *getting started* part yet, I invite you to do it now for the rest of this tutorial.

prerequisites

You'll need to have a basic `CMakeLists.txt` to able to compile your code:

```
##! Uncomment those lines if you use the gaming sdk as an external project
#if (${CMAKE_SYSTEM_NAME} STREQUAL "Linux")
#    set(LINUX TRUE)
#endif ()
```

(continues on next page)

(continued from previous page)

```
#include(FetchContent)

#FetchContent_Declare(
#         antara-gaming-sdk
#         URL https://github.com/KomodoPlatform/antara-gaming-sdk/archive/master.zip
#)

#FetchContent_MakeAvailable(antara-gaming-sdk)
#init_apple_env()

add_executable(quick_and_dirty quick_and_dirty.cpp)
target_link_libraries(quick_and_dirty PUBLIC antara::world)
```

And below a cpp file with the primitives needed to launch your game:

```
/*****************************************************************************
 * Copyright © 2013-2019 The Komodo Platform Developers.                     *
 *                                                                           *
 * See the AUTHORS, DEVELOPER-AGREEMENT and LICENSE files at                 *
 * the top-level directory of this distribution for the individual copyright  *
 * holder information and the developer policies on copyright and licensing.  *
 *                                                                           *
 * Unless otherwise agreed in a custom licensing agreement, no part of the    *
 * Komodo Platform software, including this file may be copied, modified,     *
 * propagated or distributed except according to the terms contained in the   *
 * LICENSE file                                                              *
 *                                                                           *
 * Removal or modification of this copyright notice is prohibited.            *
 *                                                                           *
 *****************************************************************************/

#include <iostream>
#include <antara/gaming/core/safe.refl.hpp>
#include <antara/gaming/world/world.app.hpp>

class example_system final : public antara::gaming::ecs::post_update_system<example_
↪system>
{
public:
    example_system(entt::registry& entity_registry) noexcept : system(entity_registry)
    {
        //! Here you can initialize your system, adding entities etc
    }

    void update() noexcept final
    {
        //! Your game logic here
        nb_iteration += 1;
        std::cout << "nb_iteration: " << nb_iteration << "\n";
        if (nb_iteration == 10ull) {
            std::cout << "Maximum iteration reached, leaving game now\n";
            this->dispatcher_.trigger<antara::gaming::event::quit_game>(0);
        }
    }

private:
```

(continues on next page)

(continued from previous page)

```
    std::size_t nb_iteration{0ull};
};

REFL_AUTO(type(example_system));

class my_world_example : public antara::gaming::world::app
{
public:
    my_world_example() noexcept
    {
        this->system_manager_.create_system<example_system>(); //! Here we load our␣
↪system to use it.
    }
};

int main()
{
    my_world_example world;
    return world.run();
}
```

1.2.4 Tutorial: Antara Gaming Systems

If you have not read the *getting started* part yet, I invite you to do it now for the rest of this tutorial.

If you have not read the ecs module documentation yet, I invite you to do it now to understand what we are doing in this tutorial series.

How to create your own system step by step ?

Setup

In this tutorial I will assume that you want to write a system for a project outside the gaming SDK. (An External Game Project)

Firstly we will need a **CMakeLists.txt**:

```
if (${CMAKE_SOURCE_DIR} STREQUAL ${CMAKE_BINARY_DIR})
    message(FATAL_ERROR "Prevented in-tree build. Please create a build directory␣
↪outside of the source code and call cmake from there")
endif ()

cmake_minimum_required(VERSION 3.14)

set(CMAKE_CXX_STANDARD 17)

project(my_game_project DESCRIPTION "my_game_description" LANGUAGES CXX)

if (NOT "${CMAKE_CXX_COMPILER_ID}" STREQUAL "Clang")
    message(FATAL_ERROR "Only Clang is supported (minimum LLVM 8.0)")
endif ()

if (${CMAKE_SYSTEM_NAME} STREQUAL "Linux")
    set(LINUX TRUE)
```

(continues on next page)

(continued from previous page)

```
endif ()

include(FetchContent)

FetchContent_Declare(
        antara-gaming-sdk
        URL https://github.com/KomodoPlatform/antara-gaming-sdk/archive/master.zip
)

FetchContent_MakeAvailable(antara-gaming-sdk)
init_apple_env()

add_executable(${PROJECT_NAME} my_example_system.cpp)
target_link_libraries(${PROJECT_NAME} PUBLIC antara::world)
```

As the `CMakeLists.txt` suggests we also need a **C++** files named `my_example_system.cpp` with the following contents:

```
#include <antara/gaming/world/world.app.hpp>

using namespace antara::gaming;

class my_world : public world::app
{
public:
    my_world() noexcept
    {

    }
};

int main()
{
    my_world world;
    return world.run();
}
```

And now we can successfully build the setup project that we just made:

> **Warning:** The project is build on OSX in the following video, if you want to build for your platform please refer to the *getting started* tutorial

Create a system

Let's create between the `using namespace` statement and the definition of the class `my_world` a system class who will be a `pre_update_system`

```
class pre_concrete_system final : public ecs::pre_update_system<pre_concrete_system>
{
public:
    //! Here the constructor can take other additional arguments but the first two␣
↪are mandatory
    pre_concrete_system(entt::registry &registry) noexcept : system(registry)
```

(continues on next page)

(continued from previous page)

```
    {

    }

    void update() noexcept final
    {
        //! Empty for the moment
    }

    ~pre_concrete_system() noexcept final = default;
};
```

Now we can load this system into our world. Place yourself at the body of the constructor of the class `my_world`.

In order to load the system we will use the function `create_system` of the `system_manager` class.

```
my_world() noexcept
{
    //! Here we don't need to add any parameters for the constructor
    //! because the mandatory parameters are forwarded by default
    this->system_manager_.create_system<pre_concrete_system>();
}
```

Now, if you compile your program and start it, you will realize that you are in an infinite loop, that's simply mean your system is running inside the game loop.

But do not panic we have a way to stop our system thanks to the dispatcher.

We will make sure that after a number of iterations from our system, we will emit a `quit_game event` that will be catched by the world and stop the gaming loop.

To do this we will create a counter as a private field of our system and increment it each time the update function is called, arrived at 10 iterations we will emit an event to leave the game

```
class pre_concrete_system final : public antara::gaming::ecs::pre_update_system<pre_
↪concrete_system>
{
public:
    //! Here the constructor can take other additional arguments but the first two␣
↪are mandatory
    pre_concrete_system(entt::registry &registry) noexcept : system(registry)
    {

    }

    void update() noexcept final
    {
        std::cout << "nb_iteration: " << (++nb_iteration) << "\n";
        if (nb_iteration == 10u) {
            this->dispatcher_.trigger<antara::gaming::event::quit_game>(0);
        }
    }

    ~pre_concrete_system() noexcept final = default;
private:
    std::size_t nb_iteration{0u};
};
```

(continues on next page)

(continued from previous page)

```
REFL_AUTO(type(pre_concrete_system)) //! This line is very important, it's give a␣
↪static reflection name function to your system, otherwise you will not compile.
```

Warning: Let's not forget the inclusion of iostream header at the top of the file.

Now, if you compile your program and start it you will quit your game after 10 iterations.

Below all the code of this tutorial:

```
#include <iostream>
#include <antara/gaming/world/world.app.hpp>

using namespace antara::gaming;

class pre_concrete_system final : public ecs::pre_update_system<pre_concrete_system>
{
public:
    //! Here the constructor can take other additional arguments but the first two␣
↪are mandatory
    pre_concrete_system(entt::registry &registry) noexcept : system(registry)
    {

    }

    void update() noexcept final
    {
        std::cout << "nb_iteration: " << (++nb_iteration) << "\n";
        if (nb_iteration == 10u) {
            this->dispatcher_.trigger<antara::gaming::event::quit_game>(0);
        }
    }

    ~pre_concrete_system() noexcept final = default;
private:
    std::size_t nb_iteration{0u};
};

REFL_AUTO(type(pre_concrete_system))

class my_world : public world::app
{
public:
    my_world() noexcept
    {
        //! Here we don't need to add any parameters for the constructor
        //! because the mandatory parameters are forwarded by default
        this->system_manager_.create_system<pre_concrete_system>();
    }
};

int main()
{
    my_world world;
    return world.run();
```

(continues on next page)

(continued from previous page)

```
}
```

Congratulations, you have managed to create your own system and add it to your game world !

1.3 Antara Gaming Modules

Table 1: Modules Tables

Name	Description	Dependencies	Header Only	Contains Script	Authors	Contributors
antara::config	Modules containing game configuration and utilities to load configuration	nlohmann_json			Roman Szterg-baum	
antara::core	Modules containing core utilities such as assets path, safe headers, etc…	EnTT refl-cpp			Roman Szterg-baum	
antara::input	Modules containing enums representing keyboard, mouse, etc…				Roman Szterg-baum	
antara::event	Module Containing all the common events that's will be used in games such as (key_pressed, key_release, etc…)	EnTT doom_meta refl-cpp antara::input antara::core			Roman Szterg-baum	
antara::timer	Module allowing us the manipulation of the game frame or retrieve the fixed delta time				Roman Szterg-baum	
antara::ecs	Module allowing us the manipulations of systems such as adding, updating, disabling, or removing systems (Contains a system manager, and system abtract class)	EnTT strong_type expected range_v3 doom_meta antara::timer antara::event antara::core			Roman Szterg-baum	
antara::scenes	Module allowing us the manipulation of scenes, possibility of adding, removing and changing a scene (Contains a Scene System Manager and an Abstract Scene class)	antara::ecs			Roman Szterg-baum	
antara::world	Module allowing the creation of application world by inheriting a simple class, contains all the necessary for the development of your video game	antara::config antara::core antara::ecs			Roman Szterg-baum	
antara::lua	System allowing the manipulation of lua scripts, attaching script to entities, creating of lua scripted systems.	Lua sol2 antara::ecs antara::core			Roman Szterg-baum	
antara::box2d	System allowing us to use box2d physics library and some usefull wrapper.	Box2D antara::ecs antara::timer antara::core			Roman Szterg-baum Tolga Ay	
antara::sfml	Systems Wrapping sfml stuffs such as audio, drawing loop, input	SFML antara::config antara::ecs antara::scenes optional(antara::lua)			Roman Szterg-baum Tolga Ay	

CHAPTER

TWO

INDICES AND TABLES

- search

INDEX

A

antara::gaming::config::load_configuration (*C++ function*), 3
antara::gaming::ecs::system_manager (*C++ class*), 4
antara::gaming::ecs::system_manager::add_system (*C++ function*), 11
antara::gaming::ecs::system_manager::clock (*C++ type*), 11
antara::gaming::ecs::system_manager::create_system (*C++ function*), 10
antara::gaming::ecs::system_manager::disable_system (*C++ function*), 10
antara::gaming::ecs::system_manager::disable_systems (*C++ function*), 10
antara::gaming::ecs::system_manager::enable_system (*C++ function*), 9
antara::gaming::ecs::system_manager::enable_systems (*C++ function*), 9
antara::gaming::ecs::system_manager::get_system (*C++ function*), 6
antara::gaming::ecs::system_manager::get_systems (*C++ function*), 7
antara::gaming::ecs::system_manager::has_system (*C++ function*), 8
antara::gaming::ecs::system_manager::has_systems (*C++ function*), 8
antara::gaming::ecs::system_manager::load_systems (*C++ function*), 10
antara::gaming::ecs::system_manager::mark_system (*C++ function*), 9
antara::gaming::ecs::system_manager::mark_systems (*C++ function*), 9
antara::gaming::ecs::system_manager::nb_systems (*C++ function*), 10
antara::gaming::ecs::system_manager::receive_add_base_system (*C++ function*), 5
antara::gaming::ecs::system_manager::start (*C++ function*), 5
antara::gaming::ecs::system_manager::system_array (*C++ type*), 4
antara::gaming::ecs::system_manager::system_manager (*C++ function*), 4
antara::gaming::ecs::system_manager::system_ptr (*C++ type*), 4
antara::gaming::ecs::system_manager::system_registr (*C++ type*), 4
antara::gaming::ecs::system_manager::update (*C++ function*), 5
antara::gaming::ecs::system_manager::update_systems (*C++ function*), 6
antara::gaming::version (*C++ function*), 4