technical desing document

SaNi Engine Core Systems

# Preface

## Version

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Description | Author(s) |
| 1.0 | 2.6.2015 | Initial version | NS & JN |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Version info

### Version 1.0

Contains initial documentation of the core system. Does not contain documentation for graphics, threading or profiling yet. These parts will be added later to this document.

## Purpose of this document

Purpose of this document is to document SaNi Engines core systems. SaNi Engine will be referenced as engine in this documentation. This document will cover every part of the core system layer.

## Use of this document

This document should be used as a reference by the developers of this project. It is good to refresh your memory once in a while, and every time the developer is uncertain of the technical structure of the system.

This document is intended to be read by software developers who are involved in developing the core system.

# Introduction

## Purpose

This document will cover every system inside the core system. These systems are:

* Primitives
* Logging
* Profiling
* Random numbers
* Assertions
* Unit testing
* Networking
* Localization
* Memory allocation and management
* Parsers
* Serialization/RTTI/Reflection
* Configuration
* Strings
* File I/O
* Math
* Threading
* Graphics wrapper

Purpose of the core is to create basis for building the game engine. These systems are required to ease the development of the engine. Also a good base layer is a must for every working engine. These systems will be described in this document in detail.

Some parts of the platform independence layer will be developed during the development of the core system. This will contain mandatory systems such as file manipulation, primitives etc.

# Platform requirements and specifications

The system has to work on Windows (from version 7 till version 10), Windows Phone (starting from version 8), iOS (versions starting from 5.1), Android (starting from versions 4.0), PlayStation 4, Xbox One and Linux systems. The system will also work inside browsers, but this platform is not on high priority at the start.

For starters the core must work on Windows, Linux, Android and Windows Phone systems. Other platforms will be added in the future when it’s possible to test the system on them. The system will be developed so that it should be easy to add another supported platform.

Reason why older Android versions aren’t supported is the market share percentage of those versions. Versions below 4.0 only have around ten percent market share. Same goes with the iOS, versions older than 5.0 hold only around two percent market share. Windows Phone is another story. Windows Phone versions older than 8 have limited market share as well and Microsoft offers new updates and versions even to older phones. Another thing is that Windows Phone versions older than 8 only support DirectX9.

# Software requirements

Windows operating system is used for developing the system. The development environment will consist of Visual Studio 2013 Ultimate (student licensing). Compiler for Windows systems will be MSVC++12. Compiler for Linux systems will be Clang. Building for Android will require NDK (native development kit) and Tegra Studio. Building for Windows Phone systems requires Windows Phone SDK.

Project files will be generated via premake4. Version control will be hosted on Github.

Jenkins will be used as automatic building service. It will run all unit tests and compiles the system for all platforms. No idea about Xbox One or PlayStation 4 platforms yet, will figure this out later. There needs to be multiple build servers to able to build for all platforms.

## External dependencies

Only external dependency of the system is unit testing library named Catch.h.

## Language specifications

System and the engine will be written with C++ and using the C++11 standard. The core system does not contain parts that are written in other language than C++.

# System Overview

The system is a collection independent modules. The system will be compiled into a static library that the upper layers of the engine will use. Platform independence layer is the lowest most layer in the engine hierarchy.



The modules of this core system will be described in this part of the document. Most important functional and structural specifications will be documented. Also every module dependencies will be described. Calling convention will be from top to down. This means that no low level modules shall not depend from higher level modules. If this does not apply during the development process, the development will be paused and redesign is required.

## Overview

This part of document covers the two most low level layers that will be written by the developers. // TODO

### Core systems

The core systems contains the most universal components. These components will be used across the engine.



### Platform independence layer

## System characteristics

## System architecture

## Classes

## Relations

# System Design

## Design methods and standards

## Documentation standards

## Naming conventions

## Programming standards

## Software development tools

## Outstanding issues