**sxcore** – основное ядро движка SkyXEngine.

Основная цель ядра обеспечение многопоточности, и предоставление некоторых базовых системных функций среди которых (основные):

* математическая основа (векторы, матрицы и прочее)
* чтение/запись файлов
* чтение/запись конфигурационных (ini) файлов
* API для работы с регистрами, для доступа данных к другим библиотекам

**Начальные функции:**

версия ядра:

SX\_LIB\_API long Core\_0GetVersion();

создание нового ядра, name - имя, is\_unic - должно ли имя ядра быть уникальным:

SX\_LIB\_API void Core\_0Create(const char\* name, bool is\_unic = true);

установка своего обработчика вывода отладочной информации:

SX\_LIB\_API void Core\_Dbg\_Set(report\_func rf);

получить имя ядра:

SX\_LIB\_API void Core\_AGetName(char\* name);

уничтожение ядра:

SX\_LIB\_API void Core\_AKill();

**Дополнительные функции:**

существует ли файл:

SX\_LIB\_API bool Core\_0FileExists(const char\* path);

копирование строки в буфер обмена, возвращаемое значение true – в случае успеха, иначе false:

SX\_LIB\_API bool Core\_0ClipBoardCopy(const char \*str);

**Многопоточность**

**Регистры данных**

Основным связующим элементом, который позволяет передавать данные в любые доступные библиотеки являются регистры данных. Они доступны в любой библиотеке/приложении использующее sxcore.

В данной версии есть несколько типов регистров данных:

bool GRegistersBool [CORE\_REGISTRY\_SIZE];

int32\_t GRegistersInt [CORE\_REGISTRY\_SIZE];

float32\_t GRegistersFloat [CORE\_REGISTRY\_SIZE];

float4x4 GRegistersMatrix [CORE\_REGISTRY\_SIZE];

float3 GRegistersFloat3 [CORE\_REGISTRY\_SIZE];

Как видно регистры данных представлены обычными массивами, размерностью CORE\_REGISTRY\_SIZE который определен в sxcore.h

В файле GRegisterIndex.h объявлены резервы номеров регистров, которые являются общими для всех библиотек.

**Функции для работы с регистрами:**

установка/получения значения из регистра bool типа:

SX\_LIB\_API void Core\_RBoolSet(int id, bool val);

SX\_LIB\_API bool Core\_RBoolGet(int id);

установка/получения значения из регистра int32\_t типа:

SX\_LIB\_API void Core\_RIntSet(int id, int32\_t val);

SX\_LIB\_API int32\_t Core\_RIntGet(int id);

установка/получения значения из регистра float32\_t типа:

SX\_LIB\_API void Core\_RFloatSet(int id, float32\_t val);

SX\_LIB\_API float32\_t Core\_RFloatGet(int id);

установка/получения значения из регистра матриц:

SX\_LIB\_API void Core\_RMatrixSet(int id, float4x4\* val);

SX\_LIB\_API void Core\_RMatrixGet(int id, float4x4\* val);

установка/получения значения из регистра float3 типа

SX\_LIB\_API void Core\_RFloat3Set(int id, float3\* val);

SX\_LIB\_API void Core\_RFloat3Get(int id, float3\* val);

**Интерфейс записи/чтения файлов ISXFile**

Режимы чтения/записи файлов:

#define CORE\_FILE\_BIN 0 //двоичный

#define CORE\_FILE\_TEXT 1 //текстовый

Функции создания экзэмпляра:

создать экземпляр класса ISXFile:

SX\_LIB\_API ISXFile\* Core\_CrFile()

открыть файл (path – абсолютный путь, type – тип текстовый/двоичный):

SX\_LIB\_API ISXFile\* Core\_OpFile(const char\* path, int type)

**ISXFile состав:**

|  |  |
| --- | --- |
| Метод | Описание |
| int Open(const char\* path, int type = CORE\_FILE\_TEXT) | открыть файл, 0 – в случае успеха, < 0 в случае провала |
| int Create(const char\* path, int type = CORE\_ FILE\_TEXT) | создать файл, 0 – в случае успеха, < 0 в случае провала |
| int Add(const char\* path, int type = CORE\_FILE\_TEXT) | добавить в конец файла, 0 – в случае успеха, < 0 в случае провала |
| size\_t ReadB(void\* dest, size\_t size) | (двоичный режим) считать в dest количество байт size, возвращаемое значение 1 – удачно считано, иное ошибка |
| size\_t WriteB(void\* src, size\_t size) | (двоичный режим) записать src в количестве size байт, возвращаемое значение 1 – удачно считано, иное ошибка |
| size\_t ReadT(const char\* format, ...) | (текстовый режим) чтение из файла, ... - только указатели, возвращает количество аргументов которым было присвоено значение |
| size\_t WriteT(const char\* format, ...) | (текстовый режим) запись в файл, возвращает количество записанных символов |
| size\_t GetSize() | размер файла в байтах |
| int ReadChar() | считать символ, сдвигая курсор на позицию вперед, возвращает целочисленное значение символа, либо CORE\_FILE\_EOF в случае конца файла |
| size\_t GetPos() | текущая позиция курсора в файле, если достигнут конец файла то вернет CORE\_FILE\_EOF |
| void SetPos(size\_t pos) | установить позицию курсора в файле |
| bool IsEOF() | достигнут ли конец файла |
| void Close() | закрытие файла |

**Интерфейс для чтения/записи конфигурационных (текстовые ini) файлов ISXLConfig**

Структура файла:

[секция]: имя родительской секции из которой данные будут вставлены в текущую секцию

ключ = значение; комментарий

Также конфигурационные файлы поддерживают Си-подобные включения:

#include <путь\_до\_файла\_относительно\_текущего отркытого>

**Предупреждение:**

*Пустая строка в файле конфигураций в самом конце обязательна, иначе не будет считан последний ключ в секции!*

*Секции и ключи хранятся в виде дерева, и нет гарантии, что может быть доступ по порядковому номеру! Однако можно получить общее количество секций/ключей, дальше плясать.*

Функции создания экзэмпляра:

создать файл экземпляр класса ISXLConfig:

SX\_LIB\_API ISXLConfig\* Core\_CrLConfig();

открыть файл конфигов (path – абсолютный путь):

SX\_LIB\_API ISXLConfig\* Core\_OpLConfig(const char\* path);

**ISXLConfig состав:**

|  |  |
| --- | --- |
| int Open(const char\* path) | открыть файл, возвращает -1 в случае провала, 0 в случае успеха |
| const char\* GetKey(const char\* section, const char\* key) | получить значения ключа key который в секции section, в случае провала возвращает 0 |
| const char\* GetKeyName(const char\* section, int key) | получить имя ключа по номеру в секции, в случае провала возвращает 0 |
| const char\* GetSectionName(int num) | получить имя секции по номеру |
| void Set(const char\* section, const char\* key, const char\* val) | установить значение val ключа key который в секции section, если чего то нет то будет создано |
| int Save() | сохранить файл, возвращает 0 в случае успеха, в случае провала значение отличное от 0 |
| int GetSectionCount() | количество секций в файле |
| int GetKeyCount() | общее количество ключей |
| int GetKeyCount(const char\* section) | общее количество ключей в секции section |
| bool SectionExists(const char\* section) | существует ли секция section |
| bool KeyExists(const char\* section, const char\* key) | существует ли ключ key в секции section |