Учебник Pawn

Учимся SAMP - скриптингу

San Andreas MULTIPLAYER version 0.3D

Пятое издание

Дата выпуска: 24 декабря 2011 года

Автор: Cloud

Официальный сайт: avalanche-net.ucoz.ru

Наконец, это свершилось! Выход очередного **пятого издания** учебника по Pawn – первого и единственного на данный момент подробного учебника по Pawn в Рунете. В очередной и в последний раз поменялось оформление учебника, и вы это уже видите, судя по обложке. Данное издание специально готовилось к выходу **SAMP 0.3D**, финальная версия, которого уже тоже, наконец, вышла.

Данный учебник уже получил положительные отзывы на форуме официального сайта и готов дальше развиваться и совершенствоваться. Прошло уже больше полгода со дня выхода первого издания (1 мая 2011 года). Несмотря на то, что на создание учебника ушло немало времени и сил, данный учебник распространяется абсолютно бесплатно.

Пятое издание по сравнению со всеми остальными самое крупное в истории развития проекта «**Учебник Pawn**». В этот раз содержание учебника кардинально переменилось и переформулированы уроки. Пришлось переписать все уроки более понятно, насколько это возможно. И очень странно, что за более полгода на форуме официального сайта не оставлено замечаний по поводу ошибок в уроках. Заранее приношу свои извинения.

Уважаемые читатели данного учебника, в том числе опытные скриптеры, убедительно прошу вас оставлять сообщения на официальном сайте по замеченным ошибкам в учебнике. В прошлом 4-ом издании учебника были найдены 3 крупных недочета, которые были исправлены в текущем издании. А также исправлено множество мелких недочетов. Все вопросы по данному учебнику оставлять на форуме официального сайта в разделе «Техническая поддержка» форум «Pawno», тема «Учебник Pawn».

Всем желающим, кто может отблагодарить меня материально, вот мои **Webmoney** и **Yandex** реквизиты:

R251373044963
Z115248837623
Яндекс деньги: 41001738240483

Буду благодарен любой разумной сумме.

Если вы желаете опубликовать данный учебник на своем сайте, зайдите на форум официального сайта **Avalanche**: avalanche-net.ucoz.ru в раздел «Техническая поддержка», форум «Pawno» в тему «Учебник Pawn». Оригинальные ссылки на файлообменики прямо из темы форума, с указанием автора учебника и его сайта обязательны. В этом же разделе и форуме, вы можете создавать темы с вашими вопросами по скриптингу Pawn.

Спасибо всем читателям, в том числе оставившим положительные отзывы о данном учебнике и проголосовавшим в опросе на официальном сайте.

Адрес электронной почты официального сайта: avalanche-net@mail.ru

Адрес электронной почты автора: cs-avalanche@mail.ru

Официальный сайт: avalanche-net.ucoz.ru

Операторы

Оператор – это литерал, который заставляет компилятор выполнять некоторое действие. Операторы воздействуют на операнды (аргументы операции). Значения, над которыми оператор проводит вычисление выражения, называются операндами. Все операторы Pawn, условно разделены на следующие категории:

- условные операторы, к которым относятся оператор условия if и оператор выбора switch;
- операторы цикла (for, while, do while);
- операторы перехода (break, continue, return) и др.

Простые операторы.

Обычные скобки являются тоже оператором, применяющимся для управления порядком вычисления выражения. Квадратные скобки применяются для индексации массива. Об операторе инкременте и декремента я расскажу вам в одном из уроков. Оператор New используется для создания переменных и массивов.

Математические операторы.

Как и в большинстве других языков программирования, Pawn поддерживает основные математические операторы: умножение (*), деление (/) сложение (+), вычитание (-) и модуль (%). Назначение первых четырех операторов вам понятно; оператор модуля формирует остаток от целочисленного деления.

Операторы отношения.

К операторам отношения, называемым операторами сравнения, относятся: «меньше» (<), «меньше или равно» (<=), «больше» (>), «больше или равно» (>=), «равно» (==) и «не равно» (!=).

Логические операторы.

Довольно часто возникает необходимость проверять не одно условное выражение, а несколько. Для проверки нескольких условных выражений существуют логические операторы: «И» (&&), «ИЛИ» (||) и «НЕ» (!)

Комментарии

Комментарии в скриптинге нужны для описания фрагмента кода. Разработчик комментирует ту или иную часть кода, для того чтобы, если код будет большим, чтобы его было легче найти. Комментарии бывают двух типов: однострочные и многострочные.

Однострочные комментарии начинается с «//» двух знаков «правый слеш». Текст, который будет располагаться, за этими знаками будет являться комментарием и в редакторе Pawno он окрасится зеленым цветом. Компилятор Pawno игнорирует комментарий. Ниже приведен пример однострочного комментария:

```
1  // Do something here
```

Многострочный комментарий начинается со знака «правый слеш» и звездочка «/*». Все строки текста после этих знаков будут также окрашены зеленым цветом. Заканчивается многострочный комментарий те ми же знаками, только наоборот «*/» - звездочка, правый слеш. Ниже приведен пример многострочного комментария:

```
1  /* Do
2  something
3  here */
```

Но сильно злоупотреблять многострочным комментарием и комментировать буквально каждую строчку кода не стоит. Многострочный комментарий можно также использовать, чтобы временно исключить из процесса компиляции компилятором Pawno закомментированный код.

Автовызываемые функции

Автовызываемые функции выполняют свои функции, в соответствии с тем как они называются.

Функция и ее параметры	Когда вызывается функция
OnGameModeInit()	Когда загрузится скрипт
OnGameModeExit()	Перед тем как скрипт завершит работу
OnPlayerRequestClass(playerid, classid)	Когда игрок выбирает класс персонажа
OnPlayerConnect(playerid)	Когда игрок подключается к серверу
OnPlayerDisconnect(playerid, reason)	Когда игрок отключается от сервера
OnPlayerSpawn(playerid)	Когда игрок появляться в своей точке возрождения
OnPlayerDeath(playerid, killerid, reason)	Когда игрок погибает или его убивают
OnVehicleSpawn(vehicleid)	Когда машина появляется в своей точке возрождения
OnVehicleDeath(vehicleid, killerid)	Когда машина взрывается
OnPlayerText(playerid, text[])	Когда игрок отправляет сообщение в чат
OnPlayerCommandText(playerid, cmdtext[])	Когда игрок отправляет команду в чат
OnPlayerEnterVehicle(playerid, vehicleid, ispassenger)	Когда игрок садится в машину
OnPlayerExitVehicle(playerid, vehicleid)	Когда игрок выходит из машины
OnPlayerStateChange(playerid, newstate, oldstate)	Когда игрок меняет свой статус
OnPlayerEnterCheckpoint(playerid)	Когда игрок встает на контрольную точку
OnPlayerLeaveCheckpoint(playerid)	Когда игрок выходит из контрольной точки
OnPlayerEnterRaceCheckpoint(playerid)	Когда игрок встает на гоночную контрольную точку
OnPlayerLeaveRaceCheckpoint(playerid)	Когда игрок выходит из гоночной контрольной точки
OnRconCommand(cmd[])	Когда игрок вводит в консоль RCON команду
OnPlayerRequestSpawn(playerid)	Когда игрок изменяет свою точку возрождения
OnObjectMoved(objectid)	Когда объект движется
OnPlayerObjectMoved(playerid, objectid)	Когда объект созданный для игрока движется
OnPlayerPickUpPickup(playerid, pickupid)	Когда игрок подбирает пикап
OnVehicleMod(playerid, vehicleid, componentid)	Когда игрок модифицирует свою машину
OnVehiclePaintjob(playerid, vehicleid, paintjobid)	Когда игрок меняет покраску машины (не просто цвет)
OnVehicleRespray(playerid, vehicleid, color1, color2)	Когда игрок меняет цвет машины
OnPlayerSelectedMenuRow(playerid, row)	Когда игрок выбирает пункт меню
OnPlayerExitedMenu(playerid)	Когда игрок выходит из меню
OnPlayerInteriorChange(playerid, newinteriorid, oldinteriorid)	Когда игрок меняет интерьер
OnPlayerKeyStateChange(playerid, newkeys, oldkeys)	Когда игрок нажимает на кнопку
OnRconLoginAttempt(ip[], password[], success)	Когда игрок пытается зайти на сервер как RCON админ.
OnPlayerUpdate(playerid)	Функция вызывается постоянно, пока игрок находится на сервере
OnPlayerStreamIn(playerid, forplayerid)	Когда игрок входит в радиус видимости другого игрока
OnPlayerStreamOut(playerid, forplayerid)	Когда игрок выходит из радиуса видимости другого игрока
OnVehicleStreamIn(vehicleid, forplayerid)	Когда машина въезжает в радиус видимости другого игрока
OnVehicleStreamOut(vehicleid, forplayerid)	Когда машина уезжает из радиуса видимости другого игрока
OnDialogResponse(playerid, dialogid, response, listitem, inputtext[])	Когда игрок нажимает на любую кнопку в диалоговом окне
OnPlayerClickPlayer(playerid, clickedplayerid, source)	Когда игрок нажимает на имя игрока в списке игроков

Ограничения сервера

Увы, в связи с ограничениями сервера, наши возможности в строительстве (расстановка объектов) и во многом другом ограничены. Ограничение на длину текста в диалоге не дает нам создавать длинные диалоги и нам приходится обходить это ограничение при помощи большого массива, что очень хорошо для сервера.

Свойства скрипта	
Максимальное количество игроков на сервере:	500
Максимальное количество транспорта в игре:	2.000
Максимальное количество моделей транспорта:	Неограниченно
Максимальное количество глобальных объектов:	400
Максимальное количество иконок на карте:	100
Максимальное количество чекпоинтов:	1
Максимальное количество гоночных чекпоинтов:	1
Максимальное количество пикапов:	2048
Максимальное количество глобальных 3D надписей:	1024
Максимальное количество 3D надписей для игроков:	1024
Максимальное количество меню:	128
Максимальное количество игровых модов:	16
Максимальное количество фильтр скриптов:	16
Максимальное число гангстерских зон:	1024
Максимальная длина текста вводимого в чат:	144
Максимальная длина имени игрока:	24
Свойства Textdraw	
Максимальное количество текстдравов:	2048
Максимальное количество отображаемых текстдравов:	92
Максимальная длина текста текстдрава:	1024
Диалоговое окна	
Максимальное количество ID диалогов:	32.000
Максимальная длина текста заголовка:	64
Максимальная длина основного текста:	2048
Максимальная длина вводимого текста:	128

Во всем остальном, конечно же, наши возможности ограничивает, во-первых: сама игра - это ID персонажей, объектов и прочих элементов игры, которых можно было бы сделать больше. И во-вторых сам язык Pawn.

RCON команды для администраторов

Rcon команды для администраторов:

/rcon login [пароль] - вход в режим администратора. Необходимо для выполнения следующих команд.

/rcon exec [имя файла конфигурации] - запустить конфигурацию сервера.

/rcon cmdlist - отобразить список всех команд.

/rcon varlist - отобразить список всех переменных.

/rcon exit - "убить" сервер.

/rcon kick [id игрока] - kick player по id. Выбрасывает игрока.

/rcon ban [id игрока] - ban player по id. Выбрасывает игрока перманентно.

/rcon banip [IP адрес] - Бан игрока по IP адресу (например /rcon banip 127.0.0.1).

/rcon unbanip [IP адрес] - Снять бан игрока (например /rcon banip 127.0.0.1).

/rcon gmx - перезапуск сценария.

/rcon changemode [мод] - смена текущей карты.

/rcon say - напечатать в чате "text" от админа.

/rcon gravity - изменяет гравитацию (стандартная гравитация 000.8).

/rcon weather - [ID погоды] изменяет погоду (Пробуйте значения от 1 до 100).

/rcon echo - напечатать текст на стороне консоли.

/rcon loadfs - загрузить filterscript (например /rcon loadfs adminfs).

/rcon unloadfs - выгрузить filterscript.

/rcon reloadfs - перезагрузить filterscript.

/rcon reloadbans - перезагрузка списка забаненых.

/rcon reloadlog - перезапуск Лога сервера.

/rcon players [имя, ID игрока] - показывает информацию о игроке (ip адрес и пинг).

/rcon password - установка или изменение пароля на сервер.

Оптимизация кода скрипта/мода

Следуйте этим примерам, тогда у вас будет меньше риск, того что ваш сервер будет тормозить, зависать или вообще вылетать.

1. Используйте в скрипте/моде как можно меньше таймеров. Каждый таймер способствует большой нагрузке сервера.
2. Функции, которые не используются в таймерах, помещайте в stock а не в public.
3. При создании массивов типа: string[256]; Подумайте заранее, какая самая большая строка может использоваться в массиве, и максимально уменьшите число в квадратных скобках.
    a. Если вы хотите вывести отформатированное функцией format сообщение в чат, максимальное количество символов в сообщении **144**. Поэтому не логично использовать string[146] и более.
    b. Максимальная длина заголовка диалогового окна – **64**. (для сведения)
    c. Максимальная длина текста внутри диалогового окна – **2048**. (для сведения)
4. Соблюдайте табуляцию/выравнивание кода (лесенка) и не используйте функций наподобие tabsize.
5. Старайтесь меньше использовать такие константы MAX_PLAYERS, MAX_VEHICLES и т.п.
6. В некоторых случаях лучше использовать функцию напрямую, чем заводить для неё переменную.
7. Если в переменной может быть значение только 0 или 1, используйте тип переменной bool.
8. Старайтесь использовать стандартные функции, в большинстве случаев они быстрее, чем их аналоги написанные сторонними скриптерами.
9. В большинстве случаев, использование **public** OnPlayerUpdate не оправдывает само себя, поэтому лучше с ним не злоупотреблять.
10. Крайне нежелательно использовать цикл в цикле, иногда встречаю такую конструкцию в некоторых скриптах.

Описание ошибок и предупреждений Pawno

Ошибки и их описание

№ ошибки	Описание на английском	Описание на русском
001	expected token: "%s", but found "%s"	пропущен символ "%s", но найден "%s"
002	only a single statement (or expression) can follow each "case"	Только один оператор или выражение может следовать после "case"
003	declaration of a local variable must appear in a compound block -;	Локальные переменные должны быть в блоке
004	function "%s" is not implemented	функция "%s" не определена
005	function may not have arguments	Функция может не иметь аргументов
006	must be assigned to an array	Это должно быть в массиве
007	operator cannot be redefined	Оператор не может использоваться
008	must be a constant expression; assumed zero	Это должно быть константой равной нулю
009	invalid array size (negative or zero)	Неправильный размер массива. Отрицательное значение или ноль
010	invalid function or declaration	Неправильная функция
011	invalid outside functions	Неправильный вывод функции
012	invalid function call, not a valid address	Неправильный вызов функции
013	no entry point (no public functions)	Не точка входа
014	invalid statement; not in switch	Неверная команда
015	"default" case must be the last case in switch statement	Оператор "default" должен быть последним
016	multiple defaults in "switch"	Несколько операторов "default" в "switch"
017	undefined symbol "%s"	неизвестный символ (неопределённая переменная) "%s"
018	initialization data exceeds declared size	несовпадение данных в массиве с указанными рамками (увеличить размер массива)
019	not a label: %s"	%s - не строка
020	invalid symbol name "%s"	ошибочное название символа (начинается с цифры, например);
021	symbol already defined: %s"	символ уже определён (дважды встречается new одного и того же символа)
022	must be lvalue (non-constant)	Должна быть левой частью
023	array assignment must be simple assignment	начения массива должны быть простыми
024	"break" or "continue" is out of context	Операторы "break" или "continue" вне границ блока
025	function heading differs from prototype	Функция заголовка отличается от прототипа
026	no matching "#if..."	"#if..." - не найдено
027	invalid character constant	Неправильные символы константы
028	invalid subscript (not an array or too many subscripts): %s"	неверное выражение, нет результата %s является недействительным массивом
029	invalid expression, assumed zero	неверное выражение, нет результата

030	compound statement not closed at the end of file	составной оператор не закрыт в конце файла, поставить return 1;} в конец мода
031	unknown directive	неверная директива (подключение плагина в дефайнах, неверное установка пути инклюда и тому подобное)
032	array index out of bounds (variable "%s")	Индекс массива превышен
033	array must be indexed (variable "%s")	Массив неизвестен
034	argument does not have a default value (argument %d)	Аргумент не имеет начального значения
035	argument type mismatch (argument %d)	Несоответствие типов аргумента
036	empty statement	Пустой оператор
037	invalid string (possibly non-terminated string)	Неправильная строка
038	extra characters on line	Лишние символы в строке
039	constant symbol has no size	Символьная константа не имеет размера
040	duplicate "case" label (value %d)	несколько раз объявлен "case" с одним и тем же параметром
041	invalid ellipsis, array size is not known	Размер массива неизвестно
042	invalid combination of class specifiers	Недопустимое сочетание класса
043	character constant exceeds range for packed string	Символьная константа превышает размер строки
044	positional parameters must precede all named parameters	
045	too many function arguments	Слишком много аргументов для функции
046	unknown array size (variable "%s")	Неизвестный размер массива %s
047	array sizes do not match, or destination array is too small	размеры массивов конфликтуют, либо целевой массив слишком маленький (нужно увеличить число в квадратных скобках)
048	array dimensions do not match	Размеры массива не совпадают
049	invalid line continuation	Неправильное продолжении линии
050	invalid range	Неправильный диапазон
051	invalid subscript, use "[ ]" operators on major dimensions	Неправильный индекс. Используйте "[ ]"
052	multi-dimensional arrays must be fully initialized	Много размерные массивы должны быть полностью установлены
053	exceeding maximum number of dimensions	Превышение максимального числа измерений
054	unmatched closing brace	Непревзойденная закрывающаяся скобка
055	start of function body without function header	Начало функции без названия
056	arrays, local variables and function arguments cannot be public (variable "%s")	
057	unfinished expression before compiler directive	Незавершенное выражение для компилятора
058	duplicate argument; same argument is passed twice	Дублирование аргумента. Аргумент передается несколько раз
059	function argument may not have a default value (variable "%s")	Аргумент не может иметь значение по-умолчанию
060	multiple "#else" directives between "#if ... #endif"	Несколько "#else" между "#if ... #endif"
061	"#elseif" directive follows an "#else" directive	"#elseif" перед "#else"
062	number of operands does not fit the operator	Количество операторов не соотвествует оператору

063	function result tag of operator "%s" must be "%s"	Результат функции %s должен быть %s
064	cannot change predefined operators	Невозможно изменить уже определенные операторы
065	function argument may only have a single tag (argument %d)	В этой функции может быть только один аргумент %d
066	function argument may not be a reference argument or an array (argument "%s")	Аргумент функции не может быть ссылкой или массивом
067	variable cannot be both a reference and an array (variable "%s")	Переменная %s не может быть как массив или ссылка
068	invalid rational number precision in #pragma	Неправильное число в #pragma
069	rational number format already defined	Формат рационального числа уже определен
070	rational number support was not enabled	Рациональное число не поддерживается
071	user-defined operator must be declared before use (function "%s")	Объявленный оператор должен быть перед использованием
072	"sizeof" operator is invalid on "function" symbols	Оператор "sizeof" не может быть использован для символов
073	function argument must be an array (argument "%s")	Аргумент %s должен быть массивом
074	#define pattern must start with an alphabetic character	#define должен начинаться с буквы
075	input line too long (after substitutions)	слишком длинная строка после интеграции параметров (возможно, много лишних пробелов, или результативные данные слишком велики для строки (по длине))
076	syntax error in the expression, or invalid function call	Неправильный синтаксис или неправильный вызов функции
077	malformed UTF-8 encoding, or corrupted file: %s	Плохая кодировка UTF-8 или плохой файл %s
078	function uses both "return" and "return "	Функция использует два "return"
079	inconsistent return types (array & non-array)	несовместимость типов возвращённых результатов (массив и немассив) (сопоставить данные в функции и изменить типы переменных)
080	unknown symbol, or not a constant symbol (symbol "%s")	Неизвестный или непостоянный символ %s
081	cannot take a tag as a default value for an indexed array parameter (symbol "%s")	Нельзя взять значение в массив %s
082	user-defined operators and native functions may not have states	Созданные пользователем функции или операторы не имеют состояния
083	a function may only belong to a single automaton (symbol "%s")	Функция может принадлежать только к одной автоматизации
084	state conflict: one of the states is already assigned to another implementation (symbol "%s")	
085	no states are defined for function "%s"	Ни одно состояние не определено для функции %s;
086	unknown automaton "%s"	Неизвестная автоматизация %s
087	unknown state "%s" for automaton "%s"	Неизвестное состояние в автоматизации;
088	number of arguments does not match definition	количество аргументов не совпадает с объявленными в функции

Фатальные ошибки и их описание

№ FATAL ошибки	Описание на английском	Описание на русском
100	cannot read from file: "%s"	невозможно прочитать/найти файл %s в стандартной директории
107	too many error messages on one line	слишком много ошибок на одной строке (обычно из-за одного неправильного параметра)

Предупреждения и их описание

№ warning	Описание на английском	Описание на русском
200	symbol "%s" is truncated to 31 characters	название переменной %s обрезается до 31 символа (укоротите название переменной %s)
201	redefinition of constant/macro (symbol "%s")	двойное определение одинаковой константы (смотреть #define)
202	number of arguments does not match definition	несовпадение количества аргументов
203	symbol is never used: "%"	символ "%" никогда не используется
204	symbol is assigned a value that is never used: "%s"	символ создан, ему ставится значение, но далее он не используется
209	function "%s" should return a value	функция %s должна возвращать какое-либо значение (return 1; к примеру)
211	possibly unintended assignment	в условии использовано не сравнение, а присвоение
213	tag mismatch	несовпадение аргументов в функции
215	expression has no effect	выражение не имеет эффекта
216	nested comment	вложенный комментарий (вынесите его за функцию)
217	loose indentation	не выровненная строка (return должен быть строго под телом функции по левому краю, либо можно добавить в начало мода строку #pragma tabsize 0, но это не рекомендуется, так как иногда может не понимать и не прочитывать скобки "{" и "}")
219	local variable "%s" shadows a variable at a preceding level	переменная дважды объявлена
224	indeterminate array size in "sizeof" expression (symbol "%s")	должен быть определён размер массива %s (если определён статиком, заменить дефайном)
225	unreachable code	невалидный код
235	public function lacks forward declaration (symbol "%s")	необходим форвард функции %s (перед функцией пишем forward(%s)

%s - имя переменной.

Стили текста

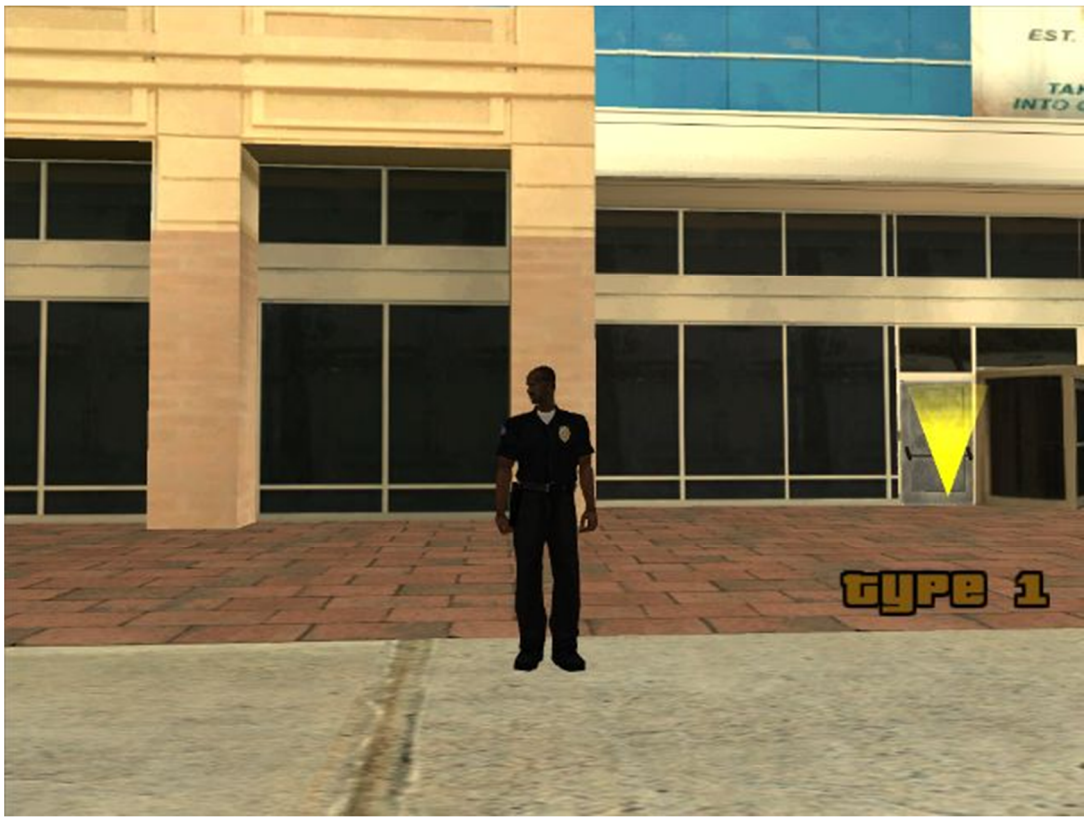
Стили текста для функций **GameTextAll** и **GameTextToPlayer**. Приведенные в таблице коды цветов используются следующим образом. Если вы хотите чтобы ваш текст был выделен определенным цветом, ставьте перед ним код цвета из таблицы. Если вы хотите, выделить определенную фразу из текста, ставьте перед началом фразы код цвета а в конце фразы код цвета основного текста.

Цвета текста:		**Пример**	**Внешний вид**
~r~	Красный	“~l~Это ~r~красный“~l~цвет ”	**Это красный цвет**
~g~	Зеленый	“~l~Это ~r~зеленый“~l~ цвет ”	**Это зеленый цвет**
~b~	Синий	“~l~Это ~r~синий“~l~ цвет ”	**Это синий цвет**
~w~	Белый	“~w~ Это белый цвет ”	**Это белый цвет**
~y~	Желтый	“~l~Это ~r~желтый“~l~ цвет ”	**Это желтый цвет**
~p~	Пурпурный	“~l~Это ~r~пурпурный“~l~ цвет ”	**Это пурпурный цвет**
~l~	Черный	“~l~Это ~r~черный“~l~ цвет ”	**Это черный цвет**
Дополнительно:			
~n~	Новая строка		

Стили игрового текста

Стиль 0

Style 1



Style 2

Style 3

Style 4

Style 5

Style 6

Вступительный урок – Знакомство с первым скриптом

Итак, давайте запустим редактор **Pawno** и создадим новый проект. В окне редактора появится ваш первый скрипт. В этом вступительном уроке мы разберем, что за что отвечает, что для чего предназначено и т.д. Итак, наш первый скрипт начинается с базового инклуда «a_samp». Без него не будет работать не один скрипт. С помощь этой директивы – include, мы во время исполнения скрипта, подключаем специальный файл к скрипту, который содержит все базовые функции Pawn.

```
1  #include <a_samp>
```

С помощью данной директивы можно подключать любой файл с расширением inc. Имя файла указывается между угловыми скобками или кавычками. Оно может быть написано как с расширением, так и без него. Данные файлы находится в директории с редактором Pawno в папке include. Файлы также могут находиться в папке include, которая находится в корневой директории сервера. Тогда, чтобы подключиться файл из данной папки, необходимо написать так:

```
1  #include <../include/a_samp>
```

Две точки означают перемещение на 1 уровень вверх. То есть мы выходим из папки с редактором Pawno в папку с сервером. Дальше идет функция main, код которой приведен ниже:

```
1  main()
2  {
3      print("\n----------------------------------------");
4      print(" Blank Gamemode by your name here");
5      print("----------------------------------------\n");
6  }
```

Функция print выводит текст, указанный в кавычках. А строковой параметр \n переводит курсор на новую строку. Следует помнить, что любой текст обязательно должен быть заключен в двойные кавычки, так как компилятор понимает это как текст.

Без всего вышеперечисленного ваш скрипт не скомпилируется. Базовый инклуд и функция main должны обязательно присутствовать в любом коде. Первые уроки и примеры, которые будут опубликованы в базовом курсе скриптинга, я буду демонстрировать их работу в консоли сервера, поскольку проще всего начинать с консоли, а потом уже переходить непосредственно в сам SAMP.

Дальше после данной функции идут все остальные функции, но на самом понятии – функция, мы остановимся более подробно на одном из уроков по создание новой функции.

Урок №01 - Переменные

Переменные – это ячейки памяти для хранения данных. Имена переменных чувствительны к регистру. В языке Pawn существует несколько типов переменные: целочисленные, вещественные и логические. Целочисленные переменные могут хранить в себе только целые числа. Целочисленная переменная или переменная целочисленного типа объявляется следующим образом:

```
1  new Pawn;
```

Таким образом, мы объявили переменную. У этой переменной нет никакого значения, поэтому она является не инициализированной. Можно задать значение этой переменной при ее создании. Присвоение переменной значения – называется инициализацией переменной, а переменная будет называться инициализированной переменной. Вот таким образом объявляется инициализированная переменная. Все аналогично и с другими типами переменных.

```
1  new Pawn = 25;
```

Вещественная переменная или переменная вещественного типа может хранить в себе только числа с плавающей точкой (десятичные числа). Переменная данного типа объявляется следующим образом:

```
1  new Float:Pawn = 25,3;
```

Таким образом, мы объявили инициализированную переменную вещественного типа со значением – 25,3. При объявлении вещественной переменной, перед именем переменной всегда пишется приставка Float:. Ну и остался последний тип переменных – логический.

Логическая переменная или переменная логического типа может хранить в себе только два логических значения «истина» - эквивалентна «1» и «ложь» - эквивалентна «0». Такой тип переменных логично использовать, если значение переменной, которую вы будете использовать, будет принимать только два значения. Объявляется переменная данного типа следующим образом:

```
1  new bool: Pawn = true;
```

Заметьте, что переменной присвоено значение «true», что значит – истина, и наоборот «false» - если ложь. Для логических переменных должны присваиваться именно такие значения: true или false. Если при объявлении переменной ее заранее не проинициализировать, то значение этой переменной будет – false. При объявлении логической переменной перед именем переменной обязательно должна быть приставка bool:.

Переменные также могут быть глобальными и локальными. Глобальные переменные, это переменные, к которым можно обратиться из любой части кода, а локальные переменные, это переменные которые находятся внутри тела функции, то есть в фигурных скобках и доступны только внутри этой функции. К локальной переменной нельзя обратиться из другой функции или части кода. Нижеприведенный пример наглядно демонстрирует, эти два типа переменных:

```
1  new Pawn; // Это глобальная переменная
2  public OnPlayerConnect(playerid)
3  {
4      new Samp; // Это локальная переменная
5      return 1;
6  }
```

Урок №02 – Функции public и stock

Функции в Pawn начинаются со слова public. Стандартные или встроенные функции, они же автовызываемые, выполняются при каком-либо игровом событии. Они так и называются, в зависимости от того, при каком условии они выполнятся. Сама функция это программный блок, который может выполниться многократно в любом месте скрипта. Каждая функция обычно возвращает какое-либо значение. Функция может возвращать только целочисленное или логическое значение и строку. Вещественное значение функция возвратить не может. Но если, функция не требует возвращать какое-либо иное значение, то она должна возвращать логическое значение: «истина» (true) – эквивалентно 1 или «ложь» (false) эквивалентно 0.

Функция состоит из имени функции, круглых скобок и фигурных скобок. Имя функции не чувствительно к регистру, как и у переменных. В круглых скобках указываются через запятую параметры функции. Число параметров функции может быть разное и может быть даже совсем без параметров. Return отвечает за возврат значения функции. Пример стандартной функции приведен ниже:

```
public OnGameModeInit()
{
    // Это тело функции
    SetGameModeText("Blank Script");
    AddPlayerClass(0, 1958.3783, 1343.1572, 15.3746, 269.1425, 0, 0, 0, 0, 0, 0);
    return 1;
}
```

Обычная функция, как например, на вышеуказанном примере, если выполнила свои задачи правильно и до конца, должна возвратить true «истина». Обратите внимание, что **SetGameModeText** то же функция и с параметром, указанным в скобках. Эта функция передает параметр, то есть текст, указанный в скобках в клиент сервера. Если вы запустите сервер, а затем клиент. В окне клиента, если там добавлен ваш сервер, вы увидите в колонке «mode» вашего сервера, этот текст.

Также можно создать свою функцию. Функция в таком случае будет называться пользовательской функцией. Создадим простейшую функцию, но перед тем как ее создать, ее сначала нужно объявить следующим образом. Пишем forward, затем через пробел имя функции и в круглых скобках параметры функции, если они не нужны, ничего не пишем, оставляем пустые скобки. Объявление нашей простейшей функции будет выглядеть следующим образом:

```
forward SayHello();
```

Объявление функции должно быть обязательно перед самой функцией. Теперь можно приступать к созданию функции.

```
public SayHello()
{
    return print(”Hello, World!”);
}
```

Print – отображает текст в консоли сервера. Это функция не будет работать, до тех пор, пока ее не вызовешь. Чтобы ее вызвать, нужно написать имя функции и в скобках ее параметры.

```
SayHello();
```

Давайте проверим наш пример в действии. Вызовем функцию из функции **OnGameModeInit**. Для этого вышеуказанный код помещаем в тело данной функции как показано на примере ниже:

```
public OnGameModeInit()
{
    SayHello(); // Вызываем функцию
    SetGameModeText("Blank Script");
    AddPlayerClass(0, 1958.3783, 1343.1572, 15.3746, 269.1425, 0, 0, 0, 0, 0, 0);
    return 1;
}
```

Теперь скомпилируем мод и запустим сервер. И в консоли мы увидим текст «Hello, world».

Итак, переделаем наш пример и создадим функцию **Say** с параметром. Назовем этот параметр **string**. Мы будем передавать в функцию строку. Строка – это массив, поэтому к имени параметра приписываем справа пустые квадратные скобки, как показано на примере ниже:

```
1  public Say(string[])
2  {
3      return print(string);
4  }
```

Теперь, чтобы вызвать функцию, и передать тот же текст в консоль, пишем следующее:

```
1  SayHello("Hello, World!");
```

Так как тело функции состоит из одной строки, эту функцию можно уместить в одну строку, как показано на примере ниже. Но, если функция будет состоять более чем из 1 строки, то фигурные скобки придется вернуть.

```
1  public Say(string[]) return print(string);
```

Если мы хотим передать какое-либо число в функцию, мы пишем просто имя параметра, если вещественное число, то приписываем к имени параметра, впереди приставку Float:, если логическое, то приставка bool:, в общем, все как у переменных:

```
1  forward Number(Float:value); // Если в функцию будет передаваться вещественное число
2  forward Number(bool:value); // Если в функцию будет передаваться логическое значение true или false
3  forward Number(value); // Если в функцию передает обычное число
```

STOCK

Создавать функцию через stock гораздо удобнее, так как это аналогичная функция, за исключением того, что ее не нужно объявлять через forward и ее желательно использовать, если за функцией не закреплен таймер.

Ну и напоследок, функции создаются для того, чтобы не повторять часто используемые фрагменты кода при выполнении скрипта в разных местах. Гораздо удобнее поместить код в функцию и вызывать из любого места. А если этот код нужно переделать, тогда это существенный плюс функции, так как не приходится переделывать код в разных местах, когда он есть уже в функции.

```
1  stock Say(string[])
2  {
3      return print(string);
4  }
```

И напоследок, наверняка просматривая стандартные функции в инклуде a_player.inc вы увидели такую в некоторых функциях перед именем параметра стоит символ – «**&**». Если мы передаем в функцию переменную, то передается не сама функция, а ее копия. По выполнению задачи функция не изменяет значение переменной, которую мы передали в функцию. Рассмотрим отличие такого способа передачи параметров.

```
1  new value = 1234;
2  stock nvalue(val)
3  {
4      val += 246;
5      printf("%d",val);
6      return 1;
7  }
```

У нас есть простейшая функция и у нас есть переменная, давайте вызовем ее в функции OnGameModeInit, чтобы посмотреть результат в консоли сервера. Но сначала, скопируйте строку 5

вышеуказанного примера после строки с вызовом функции, но измените второй параметр val на имя созданной переменной, чтоб видеть как изменится ее значение.

```
public OnGameModeInit()
{
    stock Say(value);
    printf("%d", value);
}
```

И вот что мы увидим. Две цифры в столбик: 1480 и 1234. Из этого видно, что результат функции мы увидели 1234+246=1480, а вот из второй цифры видно, что значение созданной переменной не изменелось, так как без знака & мы передаем в фунцию, только копию данной переменной, а не саму переменную. Вот в итоге мы и получаем такой результат.

Давайте теперь подставим данный символ перед именем параметра, чтобы было так:

```
new value = 1234;
stock Say(&val)
{
    val += 246;
    printf("%d",val);
    return 1;
}
```

В результате увидим две цифры: 1480 и 1480, что значит, что наша переменная изменила значение. Также можно поступить и с переменными вещественного и логического типа. Если вы передаете строку, то здесь символ & не требуется, поскольку передается именно строка а не ее копия.

Урок №03 – Константы и макросы

Define – это директива позволяющая создавать константы и макросы. Константы, это неизменяемые переменные, они сильно упрощают работу скриптерам. Константой можно заменить трудно запоминающие элементы кода, такие как, например HEX-код цвета, которым выделяется строка. В SAMP есть множество встроенных констант. Перечислять их не буду их можно увидеть в файле, a_samp.inc. Приведу пример констант:

```
1  #define COLOR_GREY 0xAFAFAFAA
2  #define COLOR_GREEN 0x33AA33AA
3  #define COLOR_RED 0xAA3333AA
4  #define COLOR_YELLOW 0xFFFF00AA
5  #define COLOR_WHITE 0xFFFFFFAA
6  #define COLOR_ORANGE 0xFF8000AA
7  #define COLOR_BRIGHTRED 0xDB0000F6
```

Эти константы можно использовать, например, в функции **SendClientMessage**, которая отправляет текстовое сообщение в чат. Теперь вместо того, чтобы писать вот так:

```
1  SendClientMessage(playerid, 0xFFFF00AA,"Hello, World");
```

Вы можете писать, вот так:

```
1  SendClientMessage(playerid,COLOR_YELLOW,"Hello, World");
```

Суть констант я думаю, теперь вам понятна, переходим к макросам. С помощью простейшего макроса можно упростить себе работу, например с таймером. Забежим немного вперед. В таймерах время указывается в миллисекундах, что не очень удобно. Чтобы писать время в секундах создадим макрос SetTimerx

```
1  #define SetTimerx (%0,%1,%2) SetTimer(%0,%1*1000,%2)
```

Но о таймерах вы узнаете в более позднем уроке, и мы все рассмотрим данный макрос на уроке по таймерам.

Теперь о значении знака процента «%» с номером. Чтобы понять их значение рассмотрим следующий пример. Многие используют не стандартную функцию **PlayerToPoint**, которая по сути не лучше стандартной **IsPlayerInRangeOfPoint**. Разница функций лишь в том, что у их первые два параметра поменялись местами, а в остальном они одинаковы и выполняют одинаковую функцию. Так вот, допустим с помощью этого макроса мы хотим превратить все функции **PlayerToPoint** в стандартные функции **IsPlayerInRangeOfPoint**.

```
1  #define PlayerToPoint(%0,%1,%2,%3,%4) IsPlayerInRangeOfPoint(%1,%0,%2,%3,%4)
```

Номер перед знаком процента идентифицирует параметр по порядковому номеру. То есть параметр playerid у функции PlayerToPoint обозначен теперь – %0 (нулем), а параметр Float:radi обозначен – %1 (единицей). Обратите внимание, первые две цифры внутри скобок и тех и других поменялись местами. Это значит, что когда вы будете использовать данный макрос, функция **PlayerToPoint** будет заменена функцией **IsPlayerInRangeOfPoint**, а первые два параметра функции поменяются местами, чтобы соответствовать синтаксису стандартной функции.

Урок №04 – Строки

Строка – это массив символов (цифр и букв). Цифра в квадратных скобок отражает количество ячеек выделенных под символы, за исключением 1 экстра-ячейки, которая должна быть всегда. Она служит для хранения идентификатора строки.

Строки используются в основном для хранения имени игрока, текста сообщения, диалога и для других целей. Стоит помнить, что не рекомендуется создавать много очень больших строк, так же как не рекомендуется создавать большие строки, которые даже не половину не используются. В строках нужно выделять ячеек приблизительно ровно столько, сколько будет использоваться. Так например имя игрока может составлять максимально 24 символа, поэтому для хранения имени игрока используйте строку с длиной не более 24 символов. Если в строке вы собираетесь хранить сообщение, которое будет отправлено игроку в чат, то строка должна быть длиной не более 144 символа, так как это ограничение на длину текста в чате. В общем длина строки зависит от ограничений сервера на текст в различных элементах скриптинга.

Строка объявляется следующим образом:

```
1  new Pawn[14] = “Hello, World!”;
```

Так создается строка или одномерный массив, который мы сразу же проинициализировали. Итак, теперь о том как получить длину строки. Для этого существует функция **strlen**, которая возвращает длину строки, в данном примере она вернет число 13. Получить размер строки с помощью этой функции можно следующим образом. Имя функции, то есть strlen, а в скобках имя строки длину которой нужно получить. В нижеприведенном примере, если скопировать эти строчки в функцию OnGameModeInit то когда мы запустим сервер, мы получим число 13 в консоли сервера.

```
1  new pawn[14] = "Hello, World!";
2  printf("%d",strlen(pawn));
```

Увы в связи с тем, что строки и обычные одномерные массивы с виду одинаковы. Мы не сможем в строке обратиться к определенной ячейке массива, то есть к определенной букве, как это делается в обычных массивах.

Урок №05 – Массивы

Раз уж дело вы прошли урок о строках, а строки являются массивами, в этом уроке я расскажу о массивах.

Массивы бывают нескольких типов, как и переменные в основном вещественные и целочисленные. О логических массивах я говорить не стану, они бессмысленные и нигде не применяются. Также они бывают одномерными, двумерными и трехмерными.

Кроме вышеперечисленных типов, существует массив, называемый **enum** - это массив переменных. Итак, начнем по порядку с одномерных массивов.

Одномерный массив

Одномерный массив объявляется в начале как обычная строка, затем следует оператор присваивания (=) и в фигурных скобках, через запятую пишутся значения ячеек. Число значений должно соответствовать числу в квадратных скобках, как это показано на примере ниже:

```
1  new pawn[5] = {347,782,632,437,721}; //одномерный массив целочисленного типа
```

В приведенном выше примере создан инициализированный массив из 5 ячеек, со своими значениями в каждой ячейке. Обращаться по ним можно по индексам ячеек: 0, 1, 2, 3, 4. Последняя экстра – ячейка – 5 не используется никогда, она нужна для хранения идентификатора массива.

Поскольку массив эта виртуальная таблица, сейчас я научу вас обращаться к определенной ячейке данной таблицы и получать из нее данные или наоборот записывать в нее данные.

Итак, чтобы обратиться к ячейке со значением 782 (смотрите массив из первого примера), вы пишете ссылку на нее. Ссылка будет выглядеть следующим образом:

```
1  pawn[1]; //ссылаемся на ячейку со значением 782
```

Не забываем, что отсчет ячеек начинается с 0, поэтому в квадратных скобках стоит единица. Теперь мы можем делать все что угодно с данным значением. Присвоим значение этой ячейке, например – 394.

```
1  pawn[1] = 394; //теперь значение в ячейке изменилось на 394
```

То же самое, можно делать и с массивом вещественного типа. Он должен содержать только десятичные числа, и объявляется он следующим образом:

```
1  new Float:pawn[5] = {347.34,782.53,632.46,437.24,721.35}; //одномерный массив вещественного типа
```

Часто одномерные массивы используют для всех игроков, то есть создают большой массив с количеством ячеек равным максимальному количеству игроков на сервере. В языке Pawn уже предусмотрена встроенная константа MAX_PLAYERS, которая равно 500. Одномерные массивы для всех игроков создаются следующим образом:

```
1  new data[MAX_PLAYERS]; //одномерный массив для всех игроков
```

То есть мы создаем огромный массив из 500 ячеек. Чтобы обратиться к своей ячейке массива мы пишем следующее:

```
1  data[playerid]; //ссылка на ячейку игрока.
```

Двумерный массив

Двумерный массив - это виртуальная таблица, состоящая из множества строк и столбцов. Первая цифра указывает количество строк, вторая цифра количество столбцов. Столбец и строка 0 тоже учитывается. Объявляется двумерный массив следующим образом:

```
new pawn[4][5] = { //двумерный массив целочисленного типа
{347,782,632,437,721},
{836,694,579,328,849},
{854,647,369,843,954},
{146,954,445,463,646}
};
```

Ну и соответственно, чтобы обратиться к ячейке, в ссылке указываете номер строки и столбца, начиная от нуля. Давайте присвоим ячейке со значением 445 новое значение:

```
pawn[3][2] = 394; //теперь значение в ячейке изменилось на 394
```

Обратите внимание, что строки в массиве помещены также в фигурные скобки, как и весь массив, они между собой разделяются запятой, как и сами ячейки массива. Последняя строка массива остается как есть без запятой. Чтобы вывести значение вышеуказанной ячейки в консоль нужно написать следующее:

```
printf("%d",pawn[3][2]); //выводим число 394 в консоль, или 445, если значения не присваивали ранее ячейке
```

Enum

Массив **enum** – это массив переменных. В этом массиве могут быть переменные различных типов. Массив объявляется следующим образом:

```
enum array
{
    pawn,
    sawn
}
```

В этом массиве объявлены две переменные. Но, чтобы обратиться к переменной в массиве вы не можете написать так:

```
array[pawn] = 10;
```

Сначала нужно создать одномерный массив, через который можно будет обращаться к переменным массива. Обычно во всех модах данный массив создается для всех игроков сразу, я рассмотрю оба варианта массива, общий и индивидуальный для каждого.

Массив создает следующим образом, создается как обычная строка, только в квадратных скобках указывается имя массива **enum**. Если создать переменную, таким образом, как показано ниже, то массив array будет общим для всех игроков.

```
new data[array]; //переменная делает массив ENUM общим для всех.
```

Если сделать массив **enum** для всех игроков, то уже нужен двумерный массив, который создается следующим образом:

```
new data[MAX_PLAYERS][array]; //переменная делает массив ENUM индивидуальным для каждого игрока.
```

То есть, переменные внутри массива **enum** – Array будут иметь индивидуальные для каждого игрока значения.

Урок №06 – Условные конструкции if-else-elseif

Условные конструкции позволяют Вам проверить, удовлетворяют ли данным условиям выражения или нет и в зависимости от результата, выполняет соответствующий код. Для того чтобы работать с условными конструкциями, сначала нужно знать операторы сравнения или условные операторы, с помощью которых ставится условие для определенной конструкции. В приведенной ниже таблице приведены все условные операторы и их применение.

Оператор	Значение	Примерное использование
&&	И	if(a && b)
||	ИЛИ	if(a || b)
!	НЕ	if(!a)
==	РАВНО	if(a==b)
!=	НЕ РАВНО	if(a!=b)
>=	БОЛЬШЕ ИЛИ РАВНО	if(a>=b)
<=	МЕНЬШЕ ИЛИ РАВНО	if(a<=b)
>	БОЛЬШЕ	if(a>b)
<	МЕНЬШЕ	if(a<b)
	НЕ ИЛИ	if(!(Left || Right))
	НЕ И	if(!(Left && Right))
	ИЛИ ИЛИ	if((Left && !Right)||(!Left && Right))
	НЕ ИЛИ ИЛИ	if(!((Left && !Right)||(!Left && Right)))

If – это условный оператор, в скобках перед оператором пишется условие. После скобок точка с запятой не ставится. В фигурных скобках пишется код который выполнится, если условие истинно. Если условие ложно, код не выполнится. **Else** – это также условный оператор, но он выполняет свои функции, только в том случае, если условие в **if** ложно. То есть, оператор **if** можно назвать как оператор «если», а **else** как оператор «иначе».

Давайте поставим простейшее условие. У нас есть две заранее проинициализированных переменных:

```
1  new a = 5;
2  new b = 25;
```

Поставим такое условие, если a < b, в консоль сервера выведется текст: «a меньше b». Если же значение переменной a не меньше b, в консоль сервера выведется текст: «a больше b».

```
1  if(a < b)
2  {
3      print("a меньше b");
4  }
5  else
6  {
7      print("a больше b");
8  }
```

В данном примере выполнится кол в **if**, так как 5 меньше 25. Но что если a равно b. Тогда мы поступим следующим образом: (между 4-5 строкой) вставим, операторы else if и условие, в общем, так как показано на примере ниже:

```
if(a < b)
{
	print(“a меньше b”);
}
else if (a == b)
{
	print(“a равно b”);
}
else
{
	print(“a больше b”);
}
```

Для некоторых случаев совсем необязателен оператор else, например, в командах чата. Например, случай такой, нам нужно, чтобы команда не выполнялась до тех пор, пока игрок не авторизуется на сервере. Мы пишем условие внутри команды, при котором команда не выполнится, и ставим в конце кода return 1. В приведенном ниже примере Return завершает работу функции **OnPlayerCommandText** и не дает ей выполниться до конца. Под данным условием уже можно будет писать код команды.

```
public OnPlayerCommandText(playerid, cmdtext[])
{
    if (strcmp("/mycommand", cmdtext, true, 10) == 0)
    {
        if (pLogged[playerid] == false)
        {
            SendClientMessage(playerid,0xAA3333AA,”Вы не авторизованы на сервере”);
            return 1; // Если это условие истинно, функция прекращает работу на этом месте
        }
        // Код команды
        return 1;
    }
    return 0;
}
```

Урок №07 – Оператор инкремента и декремента

Инкремент увеличивает значение переменной на единицу, а декремент наоборот уменьшается значение переменной на единицу. Инкременты и декременты используются в циклах, о которых я расскажу на следующем уроке. У нас есть одна из переменных из прошлого урока, мы можем ее инкрементировать или наоборот декрементировать двумя способами: с помощью **pre** и **post** инкремента или декремента.

Pre-инкремент/декремент или (префиксный инкремент/декремент) увеличивает/уменьшает значение переменной в данном примере a на единицу, а затем возвращает ее значение.

```
1  ++a;
```

Post-инкремент/декремент или (постфиксный инкремент/декремент) сначала возвращает значение переменной, а затем увеличивает/уменьшает ее значение на единицу.

```
1  a++;
```

Когда мы прибавляем к значению переменной единицу при помощи инкремента, таким образом, мы инкрементируем переменную. Когда мы отнимает от значения переменной единицу при помощи декремента, таким образом, мы декрементируем переменную.

Разницу между префиксным или постфиксным инкрементом и декрементов вы увидите в следующем примере, для этого в функции OnGameModeInit нужно добавить эти строчки и запустить сервер.

```
1  new a = 25;
2  printf("%d",a++);
```

В вышеуказанном примере показана работа постинкремента. Мы увидим число 25 а не 26, потому что постинкремент сначала возвратит нам старое значение переменой до ее инкрементирования. Если мы поставим вместо постинкремента – преинкремент, то мы увидим число 26. Аналогично и с декрементом.

Урок №08 – Циклы while и do while

Цикл While и любой другой цикл, повторяет свою функцию бесконечно до тех пор, пока его условие истинно. Если его условие будет ложно, цикл прекращает работу. Цикл выглядит следующим образом:

```
1  new a;
2  while(a <= 3)
3  {
4      a++;
5      print("Hello, World!");
6  }
```

Как это работает? Все очень просто. Цикл проверяет условие, если, а меньше или равно 5, он выполняет свою функцию. В вышеуказанном примере так оно и есть, так как наша переменная была создана, но не проинициализирована, то есть ей не задано значение и оно равно – 0. Инкрементируем переменную Post-инкрементом. То есть сначала возвращается старое значение, а потом только оно увеличивается на единицу и затем идет печать текста «Hello, World!» в консоль сервера функцией **print**. В итоге мы в консоли сервера получим четыре раза текст «Hello, World!».

Перейдем к следующему циклу, **do while**. Если **while** сначала проверял условие, а затем выполнял свою функцию, то **do while** делает все с точностью да наоборот. Сначала он выполняет свою функцию, а потом проверяет условие.

Чтобы сделать из вышеуказанного цикла, цикл do while, достаточно верхнюю часть цикла вместе с условием перенести под фигурные скобки и в конце после условия после закрывающей круглой скобки поставить точку с запятой, а на то место, откуда вы вырезали while, поставить оператор do.

```
1  new a;
2  do
3  {
4      a++;
5      print("Hello, World!");
6  }
7  while(a <= 3);
```

Тут принцип работы аналогичен, и в результате мы получаем то же самое, что и при простом цикле while.

Урок №09 – Цикл for

Цикл **For** – это, по сути, упрощенный цикл **While**, поэтому я не буду долго заострять на этом внимание. Сейчас вы убедитесь на примере из прошлого урока, который продемонстрирован ниже:

```
1  new a;
2  while(a <= 3)
3  {
4      a++;
5      print("Hello, World!");
6  }
```

Ниже приведена конструкция цикла For:

```
1  for(переменная; условие; операция с переменной-счетчиком)
2  {
3      //здесь будет код, который выполнится если условие цикла истинно;
4  }
```

Цикл for выполнять аналогичные функции, что и цикл while. Сейчас я покажу, как превратить данный цикл в цикл for. Для этого, оператор while заменяем на for. Переменную a вместе с оператором new и точкой запятой перемещаем перед условием в скобках, в конце условия также ставим точку с запятой. Берем инкремент полностью, как мы брали переменную и перемещаем после условия. А все остальное остается на месте, в результате мы получаем цикл **For**.

```
1  for(new a; a <= 3; a++;)
2  {
3      print("Hello, World!");
4  }
```

И на заметку, не рекомендуется скрипт перегружать циклами. И еще что особенно важно нежелательно использовать конструкцию: «цикл в цикле» это дает сильную нагрузку на сервер.

Но тем не менее циклы очень полезны. Их очень удобно использовать для поиска по массивам, а также для выполнения одной и той же операции для всех или определенных игроков одновременно.

Урок №10 – Операторы Break и Continue

Данные операторы используются в циклах: for, while, do-while. Оператор break завершает работу цикла, а оператор continue, пропускает оставшееся действие цикла и повторяется снова. Рассмотрим два оператора в действии. У нас есть цикл for из прошлого урока:

```
new a;
while(a <= 3)
{
	a++;
	print("Hello, World!");
}
```

Сделаем так, чтобы в консоль сервера вывелся текст «Hello World!» не 3, а 2 раза. Вспоминаем урок про условные конструкции и создаем такое условие: если а равен 3, то завершаем работу цикла.

```
new a;
while(a <= 3)
{
	if(a == 2)
	{
		break;
	}
	a++;
	print("Hello, World!");
}
```

Если вместо оператора break подставить оператор continue результат получится тот же. Вы спросите, в чем же тогда разница, если результат один и тот же. В этом цикле есть один подводный камень. Если вы прочитаете последовательность действий цикла, то поймете в чем причина. А причина вот в чем:

Цикл проверяет последовательно условия, начиная с 0, так как переменная а – равна 0. В первом и во втором случае, условия истинны, так как 0 и 1 меньше 3 и не равны 2. Соответственно в чат выводится 2 раза текст «Hello, World». Что break, что continue они не дают до конца выполнить цикл, то есть все функции, что находятся после самого оператора (инкремент и функцию print).

Для того, чтобы увидеть отличие этих двух операторов нужно инкремент переместить перед условием, вот так:

```
new a;
while(a <= 3)
{
	a++;
	if(a == 2)
	{
		break;
	}
	print("Hello, World!");
}
```

В случае с оператором **break**, в консоль выведется текст «Hello, World!» всего один раз. Так как пропускается одно условие из-за инкремента, где переменная **«a»** должна быть равна 0. В случае с оператором **continue** текст выведется три раза, так как оператор пропускает под собой все оставшиеся функции цикла и начинает следующий цикл.

Урок №11 – Форматирование строки с помощью printf и format

С помощью оператора format можно передать в строку значения переменных или функций, других строк и просто текст, который нужно передать в строку. У функции если не считать, сколько всего будет передано в строку, всего 4 аргумента. Первый аргумент, это имя строки, в которой сохранятся форматированная строка. Второй аргумент длина этой строки, обычно вместо длины строки используют функцию **sizeof** с именем строки в качестве аргумента этой функции. Затем третьим аргументом следует текст в кавычках и управляющие символы. Управляющие символы приведены в таблице ниже:

%b	Бинарный тип
%c	Символьный тип
%d, %i	Целочисленный тип
%f	Число с плавающей точкой – вещественный тип
%s	Строка

В строку можно передать неограниченное количество значений, сколько значений передается, столько и управляющих символов в строке. Главное соблюдать типы передаваемых данных. При передачи целого числа в строку, нужно использовать управляющий символ (%d) целочисленного типа, при передачи десятичного числа (%f) управляющий символ вещественного типа, при передаче другой строки нужно использовать управляющий символ - %s и так далее.

Привожу для следующего примера инициализированные строки и заранее одну пустую строку где будет сохранен форматированный текст:

```
1  new Pawn[6] = “world!”;
2  new Samp[7] = “Hello, ”;
3  new string[13];
```

Принцип работы функции такой. Он берет значение из переменной Samp указанной в нижеприведенном примере и подставляет вместо управляющего символа %s, получает текст «Hello, world!».

```
1  format(string,sizeof(string),”%s world!”,Samp);
```

Можно передать в строку сразу 2 или более значения. Для этого в 3-ем аргументе в кавычках ставим управляющий символ в то место, куда нужно будет вставить значение. Затем в зависимости от того куда мы поставили управляющий символ, в 4-ом аргументе через запятую указываем строку или переменную, из которой будет браться значение. Главное соблюдать порядок. Первая переменная или строка в 4-ом аргументе передается свое значение в 1-ый управляющий символ в 3-ем аргументе, вторая во второй, третья в третий, как показано на примере ниже:

```
1  format(string,sizeof(string),”%s %s”,Samp, Pawn);
```

Можно не указывать ни строк, ни переменных, а текст, который нам нужно вставить, вот пример:

```
1  format(string,sizeof(string),”%s” , ”Hello, world!”);
```

Все вышеуказанные примеры имеют одинаковый результат. Кроме всего вышеуказанного в строку можно вставлять следующее символы, приведенные в таблице:

\b	backspace
\f	Form feed
\n	переход на новую строку
\r	возврат каретки
\t	табуляция
\v	вертикальная табуляция
\'	одиночная кавычка
\"	двойные кавычки
\?	вопросительный знак

Функция **printf** аналогичная, она как функция format только без первого и второго аргумента, и значительно проще, так как ей не нужно никаких строк. Она выводит форматированную строку в консоль. Вот так выглядит функция, аналогичная функции на примере выше.

```
1  printf(”%s” , ”Hello, world!”);
```

Итак, для закрепления урока рассмотрим нижеприведенный пример:

```
1  public OnPlayerConnect(playerid)
2  {
3      new plname[MAX_PLAYER_NAME];
4      new string[50];
5      GetPlayerName(playerid,plname,24);
6      format(string,sizeof(string),"*** %s зашел на сервер. (ID:%d)",plname,playerid);
7      SendClientMessageToAll(COLOR_GREEN,string);
8      return 1;
9  }
```

Для того чтобы вывести сообщение всем игрокам, существует функция **SendClientMessageToAll**. В скобках функции два аргумента. Первый аргумент, цвет сообщения, куда мы подставляем HEX-код цвета либо заменяем его константой для удобства, второй аргумент – текст, выводимый в чат всем игрокам.

Итак, для того чтобы вывести сообщение всем игрокам, для начала нам нужно 2 строки (одномерных массива). MAX_PLAYER_NAME – это встроенная константа, она равна 24, то есть максимальной длине имени игрока. Первая строка будет хранить имя игрока, вторая будет хранить отформатированный текст сообщения, передаваемый игрокам. Функция **GetPlayerName** получает имя игрока, и как показано во 2 аргументе функции, передает в переменную **plname** это имя.

Имя нам известно, ID игрока тоже, теперь пора приступить к форматированию строки. Пишем функцию **format** и первым аргументом функции указываем переменную **string**, где будет храниться сообщение. Вторым аргументом при помощи функции **sizeof**, передаем размер этой переменной. В третьем аргументе функции пишем текст. Имя, это строка, для передачи в строку, нужен управляющий символ «строка» (%s), ID игрока, это целое число, для передачи в строку, нужен управляющий символ целочисленного типа (%d).

```
1  format(string,sizeof(string),"*** %s зашел на сервер. (ID:%d)",plname,playerid); // Это правильно
```

Функция по порядку передаст значения между 3 и 4 аргументами функции. Из переменной **plname** в первый управляющий символ в строке (%s), ID игрока мы передаем из параметра **playerid** во второй управляющий символ (%d).

```
1  format(string,sizeof(string),"*** %s зашел на сервер. (ID:%d)",playerid,plname); // Это не правильно
```

Выше показан заведомо неправильный пример. Мы не можем передать ID игрока то есть целое число в управляющий символ «строка», так как число это не текст (не строка). Имя игрока мы также не можем передать в управляющий символ целочисленного типа, так как строка (или текст) это не число. При форматировании строки соблюдайте типы передеваемых данных и управляющие символы для этих типов.

Урок №12 – Арифметические выражения в Pawn

Переменные целочисленного и вещественного типа, могут принимать различные числовые значения. Со значениями переменных можно проводить различные математические операции. Для этого есть операторы: сложения, вычитания, деления и умножения, которые приведены в таблице ниже:

Оператор	Обозначение	Примерное использование
Оператор сложения	+	result = pawn + samp;
Оператор вычитания	-	result = pawn - samp;
Оператор умножения	*	result = pawn * samp;
Оператор деления	/	result = pawn / samp;
Оператор деления с остатком	%	result = pawn % samp;

Примеры, указанные в таблице выше четко отображают простейшие математические операции между значениями двух переменных pawn и samp. Давайте рассмотрим данные примеры на переменных, которые приведены ниже:

```
1  new pawn = 24;
2  new samp = 6 ;
3  new result;
```

Переменная, которая должна присвоить себе результат арифметического выражения всегда должна быть впереди выражения. С помощью скобок в арифметических выражениях можно задать последовательность выполнения вычислений.

```
1  result = pawn + samp; //result = 30
2  result = pawn - samp; //result = 18
3  result = pawn * samp; //result = 144
4  result = pawn / samp; //result = 4
```

Ну и напоследок о делении с остатком, в нижеприведенном примере показан пример деления с остатком:

```
1  result = (pawn + samp) % 7; //result = 2
```

Как получилось 2? Все просто, складываем значения переменных, так как они в скобках и делим на 7. Мы получаем приблизительно 4,2. Если умножим 4 на 7, получим 28, ближайшее к 30 число. Если отнимем 30 от 28, получим тот самый остаток.

Решим еще одно выражение, пусть переменная pawn будет равна 30. В итоге сумму переменных то есть 36 делим на 7 и получаем 5. Умножаем 5 на 7, получаем 35. Отнимаем его из 36, в результате чего получаем тот остаток 1.

Урок №13 – Оператор switch

Оператор switch – это оператор выбора. Он удобен в первую очередь тем, что может заменить много условий if, которые проверяют значение одной переменной. Представим себе такой пример. У нас есть целочисленная переменная pawn, значение которой нужно проверить. И в зависимости от того, какое значение у данной переменной выполнить соответствующий код.

```
1  new pawn;
2  switch(samp)
3  {
4      case 0:  //аналог условия if(samp == 0)
5      {
6          print("Переменная samp = 0");
7      }
8      case 1: //аналог условия if(samp == 1)
9      {
10         print("Переменная samp = 1");
11     }
12 }
```

Обратите внимание, если в фигурных скобках указано одно действие, его можно упростить так же как в случае с обычной условной конструкцией.

Но, что если переменная не попадает ни под одно условие в операторе switch. Тогда можно использовать **default**, он выполняется только в том случае, если **switch** не попадает ни в один из **case**.

```
1  new pawn;
2  switch(samp)
3  {
4      case 0:  printf("Переменная samp = 0");
5      case 1: printf("Переменная samp = 1" );
6      default: printf("Переменная samp = %d" , samp);
7  }
```

То же самое можно сделать и с помощью обычной условной конструкции if, тогда код выглядел бы следующим образом:

```
1  if(samp == 0)
2  {
3      printf("Переменная samp = 0"); //все равно что case 0
4  }
5  else if(samp == 1)
6  {
7      printf("Переменная samp = 1"); //все равно что case 1
8  }
9  else
10 {
11     printf("Переменная samp = %d" , samp); //все равно что default
12 }
```

Этот способ удобнее и читабельнее чем много if конструкций идущих подряд. Еще одно отличие, что в case можно указать диапазон значений куда проще, чем в обычной условной конструкции if.

```
1  case 0..2:  printf("Переменная samp = %d" , samp); //условие внутри switch
2  if(samp > 0 && samp < 2) printf("Переменная samp = %d" , samp); //то же условие, но уже в if
```

Для закрепления знаний по данному урока продемонстрирую вам один из примеров использования данного оператора в функции OnPlayerDisconnect. В данной функции параметр reason возвращает ID причины отключения игрока от сервера. Благодаря этому параметру можно сделать вот такой код оповещения игроков о уходе игрока с сервера с пояснением причины.

```
switch(reason)
{
    case 0:
    {
        GetPlayerName(playerid,plname,24);
        format(string,sizeof(string), "*** %s вылетел с сервера.(ID:%d)",plname,playerid);
        SendClientMessageToAll(0xAA3333AA,string);
    }
    case 1:
    {
        GetPlayerName(playerid,plname,24);
        format(string,sizeof(string), "*** %s покинул сервер.(ID:%d)",plname,playerid);
        SendClientMessageToAll(0xAA3333AA,string);
    }
    case 2:
    {
        GetPlayerName(playerid,plname,24);
        format(string,sizeof(string), "*** %s был кикнут.(ID:%d)",plname,playerid);
        SendClientMessageToAll(0xAA3333AA,string);
    }
}
```

Этот пример можно упростить с помощью вот такой **stock** функции.

```
stock GetName(playerid)
{
    new nick[MAX_PLAYER_NAME];
    GetPlayerName(playerid, nick, sizeof(nick));
    return nick;
}
```

В этом случае функцию **GetPlayerName** можно просто убрать из **switch**, а вместо переменной **plname** подставить вызов данной функции. Саму переменную **plname** также нужно удалить.

Урок №14 – Создание простейшей команды

Функция **OnPlayerCommandText**, выполняется, в том случае если игрок введет команду в чат. Первый параметр функции, это ID игрока, который ввел команду, второй параметр, это строка, которая хранит введённую игроком команду. Чтобы создать простейшую команду, нам нужно проверить правильно ли ввел ее игрок. Для этого нам надо сравнить введенную игроком команду с ее оригиналом.

Для сравнивания строк существует функция **strcmp**.

```
1  public OnPlayerCommandText(playerid, cmdtext[])
2  {
3      if(strcmp("/mycommand", cmdtext, true, 10) == 0)
4      {
5          //код команды
6          return 1;
7      }
8  }
```

strcmp – имеет четыре аргумента. Первый аргумент имя строки, которая будет сравниваться с другой строкой, второй аргумент – имя строки, с которой будет сравниваться предыдущая строка. Третий аргумент логический аргумент. Если указать true – то при сравнении строк регистр букв не учитывается, по умолчанию, если даже его не указать регистр учитываться будет.

Хочу отметить, что если вы не указали третий аргумент, то и четвертый аргумент уже указать не получится, поскольку пустой аргумент оставлять нельзя. Также если третий аргумент не обязательный для указания, а четвёртый аргумент обязательный, то третий аргумент, так или иначе, придется указать, пропускать аргументы нельзя. Так как у функции **strcmp** два последних аргумента не обязательны, их можно не указывать, поскольку они последние. Сама функция возвращает 0 – если совпадение 100%, в противном случае возвратит 1.

Ну а дальше все зависит от вас. Если условие по сравнению команд выполняется, команда обязательно должна возвратить true (return 1;), потому что функция **OnPlayerCommandText** всегда возвращает false. Если функция возвратит false, в чат будет выведено сообщение «Неизвестная команда», а сама команда сработает.

Урок №15 – Перебор значений в массиве

Что если нужно узнать, если определенное число в массиве или нет. Но функции поиска по массиву в Pawn нет. Что делать? Поможет только перебор значений. Для этого нам нужен сам массив и цикл for. Вы уже знакомы с массивами и циклами из прошлых уроков. Сейчас я на простом примере покажу, как сделать перебор значений в массиве. И при появлении совпадения, должен выполниться скрипт.

Допустим, у нас есть большой одномерный массив со всеми ID машин, т. е только легкового транспорта. Но сначала нам нужна новая автовызываемая функция, допустим, она будет называться IsPlayerInBike. В скобках пишем два параметра – playerid и vehicleid. Функция будет выглядеть следующим образом:

```
1  stock IsPlayerInBike(playerid,vehicleid)
2  {
3          return 0;
4  }
```

Поскольку функция не нуждается в таймере, то есть ее не нужно выполнять по истечению определенного количества времени или повторять, то лучше сделать эту функцию не через **public**, а через **stock**. Функция должна возвращать 0, потому что мы делаем функцию проверки и соответственно функция должна возвращать либо true, либо false. Внутри функции мы добавим одномерный массив.

```
1  stock IsPlayerInBike(playerid,vehicleid)
2  {
3          new IsBikeA[10] = {581,523,462,521,463,522,461,448,468,586};
4          return 0;
5  }
```

Теперь нам нужно присвоить второму аргументу нашей функции vehicleid - ID транспорта, в котором сидит игрок. Чтобы узнать ID машины, существует функция **GetPlayerVehicleID**, которая возвращает ID транспорта, в котором сидит игрок. Для этого мы присвоим аргументу значение, которое возвратит нам вышеуказанная функция. Делаем это следующим образом, под массивом мы добавляем следующую строку:

```
1  vehicleid = GetPlayerVehicleID(playerid);
```

Дальше нам нужно проверить сидит ли игрок в машине с указанным ID. Функция **IsPlayerInVehicle**, проверяет: сидит ли игрок в транспорте с указанным во 2 аргументе функции ID транспорта. На следующей строчке пишем следующую условную конструкцию;

```
1  if(IsPlayerInVehicle(playerid,vehicleid))
2  {
3
4  }
```

В теле этой условной конструкции будет располагаться цикл, который будет искать требуемое значение в массиве. Условие цикла должно быть следующее, если I < 10. Переменная I у нас, это переменная счетчик, как в любом цикле. То есть мы должны выполнить цикл 10 раз, так как в массиве у нас всего 10 ячеек или ID 10 мотоциклов.

```
1  for(new i = 0; i < 10; i++)
2  {
3    if(GetVehicleModel(vehicleid) == IsBikeA[i]) return true;
4  }
```

Если совпадение найдено, функция должна вернуть true, если нет false. Под циклом обязательно должен быть return 0, так как если совпадений не найдется, функция возвратит false.

Урок №16 – Расстановка транспорта

Расставить транспорт можно несколькими способами:

Расстановка транспорта с помощью **Samp Debug**:

Для начала запускаем **Samp Debug** из папки с GTA San Andreas. Как только все загрузится, вы появляетесь Карлом Джонсоном возле надписи Vinewood.

Теперь с помощью команды **/v** вызываем нужную нам машину, например INFERNUS, для этого пишем в чате /v 411 и получаем машину. Если вы не знаете ID машины, которую хотите вызвать, введите **/vmenu**, и вы попадете в меню выбора транспорта.

Садитесь в машину и паркуйте ее там, где вам нужно. После этого, не выходя из машины, вводите команду **/save**. В директории: Мои документы\GTA San Andreas User Files\Samp\ будет файл savedpositions.txt, с примерно таким содержанием:

```
1 AddStaticVehicle(411, 322.5014, 303.6906, 999.1484, 269.1425, 0, 0);
```

Функция **AddStaticVehicle** может использоваться только внутри автовызываемой функции **OnGameModeInit**, в других функциях она работать не будет.

Можно расставить транспорт с помощью фильтрскриптов, таких как cars. Данный скрипт прилагается к учебнику. Достаточно ввести ID транспорта, как команда, например /411 и вы окажетесь прямо в машине.

Также для расстановки транспорта существует функция **AddStaticVehicleEx**, большое отличие данной функций от вышеуказанной в том, что данная функция возвращает ID созданного транспорта, а вышеуказанная функция ничего не возвращает. Это расширенная функция.

```
1 AddStaticVehicle(411, 322.5014, 303.6906, 999.1484, 269.1425, 0, 0, 30000);
```

Последняя цифра в функции указывает на время в миллисекундах до следующего появления транспорта на точке возрождения (spawn).

Урок №17 – Проверка игрока в зоне

Чтобы проверить игрока в зоне, нам нужно написать в любом месте скрипта такую функцию:

```
stock IsPlayerInArea(playerid, Float:minx, Float:miny, Float:maxx, Float:maxy)
{
    new Float:X, Float:Y, Float:Z;
    GetPlayerPos(playerid, X, Y, Z);
    if(X >= minx && X <= maxx && Y >= miny && Y <= maxy)
    {
        return 1;
    }
    return 0;
}
```

Теперь мы можем ставить такие условия:

```
if(IsPlayerInArea(playerid,-36.5483,-57.9948,-17.2655,-49.2967))
```

Последняя цифра в функции указывает на время в миллисекундах до следующего появления транспорта на точке возрождения (**spawn**). Эту проверку можно поместить, например, внутри команды.

Теперь о том, как правильно получить все 4 координаты, указанные в скобках функции. Синтаксис условия такой:

```
if(IsPlayerInArea(playerid,Xmin,Ymin,Xmax,Ymax))
```

Итак, встаем в левый нижний угол создаваемой вами зоны, получаем координаты командой save. Далее встаем в правый верхний угол и снова получаем координаты. Допустим, мы уже получили вот такие координаты:

```
AddPlayerClass(0,2021.0109,1343.0779,10.8130,256.6816,0,0,0,0,0,0);
AddPlayerClass(0,2038.6593,1343.9640,10.3990,180.1545,0,0,0,0,0,0);
```

Нужные нам координаты, я выделил цветом, т.е. это все координаты X и Y. Берем из этих координат минимальную X-координату - 2021.0109. Ставим ее первой в скобках, затем через запятую минимальную Y-координату - 1343.0779. Дальше то же самое, но уже максимальные координаты. Должно получиться так:

```
if(IsPlayerInArea(playerid,2021.0109,1343.0779, 2038.6593,1343.9640))
```

Урок №18 – Проверка игрока в кубе

Чтобы проверить игрока в кубе, нам нужно написать в любом месте скрипта такую функцию:

```
stock IsPlayerInCube(playerid, Float:xmin, Float:ymin, Float:zmin, Float:xmax, Float:ymax, Float:zmax)
{
    new Float:x, Float:y, Float:z;
    GetPlayerPos(playerid, x, y, z);
    if(x > xmin && y > ymin && z > zmin && x < xmax && y < ymax && z < zmax)return 1;
    return 0;
}
```

Теперь мы можем ставить такие условия:

```
if(IsPlayerInCube(playerid, 72.1256, 1544.2145, 15.7742, 75.2350, 1546.3352, 16.4206))
```

Эту проверку можно поместить, например, внутри команды. Теперь о том, как правильно получить все 4 координаты, указанные в скобках функции. Синтаксис условия такой:

```
if(IsPlayerInCube (playerid,Xmin,Ymin,Zmin,Xmax,Ymax,Zmax))
```

Итак, встаем в левый нижний угол создаваемой вами зоны, получаем координаты командой save. Далее встаем в правый верхний угол и снова получаем координаты. Допустим, мы уже получили вот такие координаты:

```
AddPlayerClass(0,2021.0109,1343.0779,10.8130,256.6816,0,0,0,0,0,0);
AddPlayerClass(0,2038.6593,1343.9640,10.3990,180.1545,0,0,0,0,0,0);
```

Нужные нам координаты, я выделил цветом, т.е. это все координаты X и Y. Берем из этих координат минимальную X-координату - 2021.0109. Ставим ее первой в скобках, затем через запятую минимальную Y-координату - 1343.0779. Дальше то же самое, но уже максимальные координаты. Должно получиться так:

```
if(IsPlayerInCube(playerid,2021.0109,1343.0779,10.3990, 2038.6593, 1343.9640,10.8130))
```

Урок №19 – Проверка игрока в радиусе

Чтобы проверить игрока в радиусе, нам нужно написать в любом месте скрипта такую функцию:

```
stock PlayerToPoint(Float:radi, playerid, Float:x, Float:y, Float:z)
{
    if(IsPlayerConnected(playerid))
    {
        new Float:oldposx, Float:oldposy, Float:oldposz;
        new Float:tempposx, Float:tempposy, Float:tempposz;
        GetPlayerPos(playerid, oldposx, oldposy, oldposz);
        tempposx = (oldposx -x);
        tempposy = (oldposy -y);
        tempposz = (oldposz -z);
        if (((tempposx < radi) && (tempposx > -radi)) && ((tempposy < radi) && (tempposy > -radi)) && ((tempposz < radi) && (tempposz > -radi)))
        {
            return 1;
        }
    }
    return 0;
}
```

В отличие от проверки в зоне или кубе тут достаточно получить 1 раз координаты и вручную вписать радиус. Написав автовызываемую функцию **PlayerToPoint**, мы можем ставить такие условия:

```
if(PlayerToPoint(3.0,playerid, 72.1256, 1544.2145, 15.7742))
```

Эту проверку можно поместить, например, внутри команды. Но в целях оптимизации рекомендуют использовать стандартную функцию под названием **IsPlayerInRangeOfPoint**, это аналог данной функции, в котором только первый и второй параметры переставлены местами. Синтаксис функции такой:

```
IsPlayerInRangeOfPoint(playerid, Float:range, Float:x, Float:y, Float:z);
```

Соответственно проверка при использовании данной функции выглядела бы таким образом:

```
if(IsPlayerInRangeOfPoint(playerid, 3.0, 72.1256, 1544.2145, 15.7742))
```

Урок №20 – Создание простого диалога

В дальнейших уроках мы рассмотрим все типы диалогов и работу с ними. Для начала я ознакомлю вас со всеми диалогами, а затем в каждом уроке мы будем рассматривать по каждому типу диалога. Итак, приступим к теории. Диалоги бывают трех типов:

0 - DIALOG_STYLE_MSGBOX - обычный диалог с 2мя кнопками,
1 - DIALOG_STYLE_INPUT - диалог с полем для ввода,
2 - DIALOG_STYLE_LIST - список из нескольких элементов.
3 - DIALOG_STYLE_PASSWORD – диалог с полем ввода пароля.

Вызывается диалоговое окно функцией **ShowPlayerDialog**, структура функции такая:

```
1  ShowPlayerDialog(playerid,<ID диалога>,<ID стиля>,<Название>,<Текст>,<Первая кнопка>,<Вторая>);
```

Обычно, чтобы не запоминать цифры и не писать длинное название стилей диалогов, я заменяю их такими константами:

```
1  #define DSL DIALOG_STYLE_LIST
2  #define DSI DIALOG_STYLE_INPUT
3  #define DSM DIALOG_STYLE_MSGBOX
4  #define DSP DIALOG_STYLE_PASSWORD
```

Для написания основного текста диалогового окна вы можете использовать нижеприведенную таблицу:

\b	backspace
\f	Form feed
\n	переход на новую строку
\r	возврат каретки
\t	табуляция
\v	вертикальная табуляция
\'	одиночная кавычка
\"	двойные кавычки
\?	вопросительный знак

Итак, переходим к практике или к разбору первого стиля: **DIALOG_STYLE_MSGBOX**. Рассмотрим обычный пример: Мы вводим команду, и появляется диалоговое окно с запросом о подтверждении выполнения команды. Для начала мы напишем скрипт простейшей команды в автовызываемую функцию **OnPlayerCommandText**.

```
1  if (strcmp("/test", cmdtext, true, 10) == 0)
2  {
3  return 1;
4  }
```

Затем с помощью функции **ShowPlayerDialog** мы вызовем данное диалоговое окно. Так как вызывать меню мы будем с помощью команды, то и пишем данную функцию, приведенную ниже внутри нее.

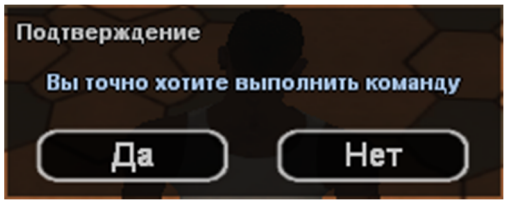
```
1  ShowPlayerDialog(playerid,0,DSM,"Подтверждение","Вы точно хотите выполнить команду","Да","Нет");
```

Цифра 0 – это идентификатор (ID) диалога, у каждого диалога свой ID. После идентификатора пишем название стиля диалога, но я заменил его константой, поэтому я написал DSM (DIALOG_STYLE_MSGBOX). После стиля пишем название диалога. Далее пишем текст диалога и название первой и второй кнопки.

В общем, код будет выглядеть следующим образом:

```
if (strcmp("/test", cmdtext, true, 10) == 0)
{
ShowPlayerDialog(playerid,0,DSM,"Подтверждение", "Вы точно хотите выполнить команду","Да","Нет");
return 1;
}
```

А диалоговое окно будет выглядеть так:



Но данное диалоговое окно работать не будет, если мы не напишем для него функцию. Допустим, наша команда выдаст в чат всю информацию об игроке, т.е. при нажатии кнопки «Да» все так и будет, но при нажатии кнопки «Нет», мы выдадим сообщение красным цветом о том, что данная команда не была выполнена.

Писать функцию диалогового окна мы будем в автовызываемой функции **OnDialogResponse**. Сначала внутри функции мы напишем условную конструкцию, которая будет проверять, что данное диалоговое окно было вызвано. Код условия такой:

```
if(dialogid == 0) //Условие: если мы вызвали диалог с ID = 0
{

}
```

Условие проверяет: было ли вызвано диалоговое окно с ID = 0. Внутрь этой условной конструкции добавляем еще одно условие:

```
if(response) //Условие: если мы нажали первую кнопку
{

}
else //Условие: если мы нажали вторую кнопку
{

}
```

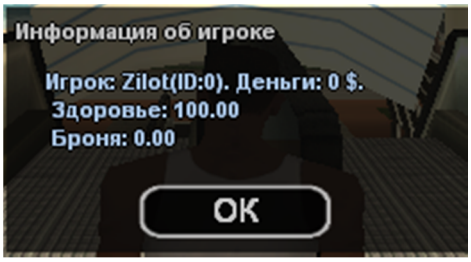
Условие проверяет, была ли нажата первая кнопка, если нажата вторая кнопка, то выполняется код после else. Внутри этой условной конструкции для первой кнопки мы пишем следующий код.

```
if(response)
{
new string[128]; //Переменная с основным текстом диалога
new plname[MAX_PLAYER_NAME]; //Переменная с именем игрока
new money; //Переменные для хранения кол-ва наличных денег игрока
new Float: health, Float: armor; //Переменные для хранения кол-ва здоровья и брони игрока
GetPlayerName(playerid,plname,MAX_PLAYER_NAME); //Записываем имя игрока в переменную plname
money = GetPlayerMoney(playerid); //Записываем деньги игрока в переменную money и т.д.
GetPlayerHealth(playerid,health);
GetPlayerArmour(playerid,armor);
format(string,sizeof(string),"Игрок: %s(ID:%d). Деньги: %d $.\n Здоровье: %.2f\n Броня: %.2f",
plname,playerid,money,health,armor);
ShowPlayerDialog(playerid,1,DSM,"Информация об игроке",string,"ОК","");
}else{}
```

Сначала мы создаем все необходимые переменные, а потом получаем данные и сразу же их записываем их во всех переменные. Обратите внимание на функцию **GetPlayerMoney**, в скобках она имеет всего один аргумент (playerid), в отличие от других, и она возвращает количество денег игрока. Поэтому мы присваиваем переменной money, эту функцию, которая возвратит значение, которое и будет значением этой переменной.

Еще обратите внимание на **%.2f**, я мог бы использовать просто **%f**, как показано в таблице. Точка и цифра означает, сколько чисел должно быть после запятой у десятичного числа. Дальше мы форматируем сообщение, после чего вызываем новое диалоговое окно с новым ID. Обратите внимание, что имя второй кнопки не указано, в таком случае, эта кнопка просто будет отсутствовать в диалоге. А вместо основного текста мы подставляем переменную string в которой уже есть готовое отформатированное сообщение.

Второе диалоговое окно будет выглядеть следующим образом:



Вернемся к условию, если будет нажата вторая кнопка в первом диалоге, тут все просто без объяснений в else пишем:

```
1  SendClientMessage(playerid,COLOR_RED,"Команда отменена");
```

Вот в принципе и все, В следующем уроке мы рассмотрим другой стиль диалога. Переходим к следующему уроку.

Урок №21 – Создание простого диалога с полем ввода

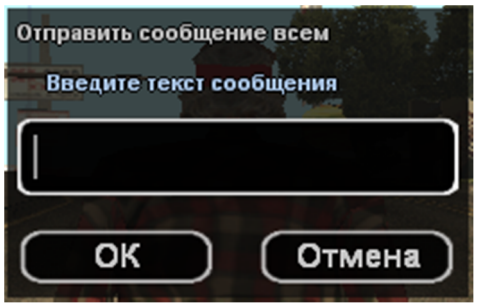
В этом уроке мы переходим к разбору второго стиля диалога: **DIALOG_STYLE_INPUT**. Работать с данным стилем диалога будет сложнее по сравнению с остальными стилями диалогов, т.к. тут есть поле ввода. Начнем с простого примера: мы командой вызовем диалоговое окно, введем текст и данный текст отправиться всем игрокам в чат. Имя игрока видно не будет, сообщение анонимное. Итак, приступим к созданию диалога:

```
1  ShowPlayerDialog(playerid,0,DSI,"Отправить сообщение всем","Введите текст сообщения","ОК","Отмена");
```

Данную строку я думаю, вы уже догадались, что ее нужно вписать внутрь готовой команды. Я все также использовал константу DSI из прошлого урока, чтобы не писать длинное название стиля диалога, поэтому не забудьте ее добавить в ваш скрипт.

```
1  #define DSI DIALOG_STYLE_INPUT
```

Итак, диалоговое окно мы создали, и оно будет выглядеть следующим образом.



Приступим к написанию функции для этого диалога. Как и в прошлом уроке, внутрь функции **OnDialogResponse**, пишем условную конструкцию проверки на вызов диалога с ID = 0. Внутрь этой условной конструкции пишем еще одну, проверку на нажатие кнопки. Дальше внутри этой проверки будет еще одна проверка на то, что поле ввода у нас содержит какие-либо символы. Ведь мы же не можем отправить пустой текст. Условная конструкция будет такая:

```
1  if(!strlen(inputtext))
2  {
3
4  }
5  else
6  {
7
8  }
```

Данную условную конструкцию мы пишем внутри условной конструкции if(response). А теперь я вам объясню, что значит strlen и inputtext:

Strlen – возвращает количество символов в строке.

Inputtext – этот аргумент содержит в себе текст, который мы вводим в поле ввода.

В общем, думаю, вам будет все понятно, посмотрев нижеприведенный готовый скрипт, но все же я вам кратко его объясню на всякий случай. Сначала идет проверка на вызов диалогового окна с ID = 0. Мы данное окно вызывали функцией **ShowPlayerDailog**(playerid,**0**…); - я специально выделил цифру, это dialogid. После проверки, внутри идет еще одна проверка на нажатие кнопки. Если нажата кнопка 1, т.е. кнопка «ОК», идет следующая проверка на то, что в поле ввода есть какой-либо текст. Если он есть, функция **SendClientMessage** отправляет в чат значение аргумента inputtext, т.е то, что игрок введет в поле ввода диалогового окна. Если игрок не введет в поле ввода ничего, функция

SendClientMessage отправит в чат сообщение с ошибкой, о том, что текст сообщения не был введен в поле ввода.

```
public OnDialogResponse(playerid, dialogid, response, listitem, inputtext[])
{
    if(dialogid == 0)
    {
        if(response) //Если была нажата 1 кнопка
        {
            if(!strlen(inputtext)) //Если в поле не был введен текст при отправке
            {
                SendClientMessage(playerid,COLOR_RED,"Вы не написали текст сообщения в поле ввода");
                return 1;
            }
            else
            {
                SendClientMessage(playerid,COLOR_WHITE,inputtext);
            }
        }
    }
    return 1;
}
```

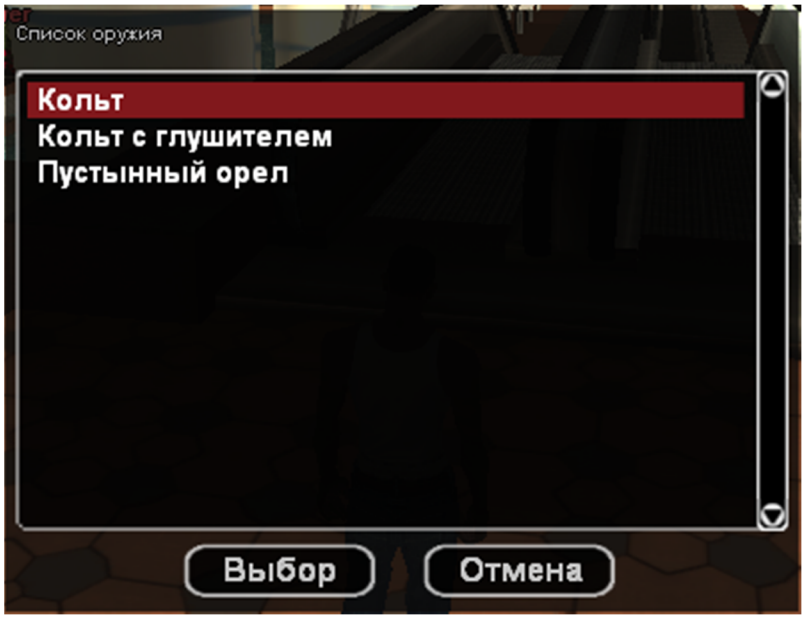
Урок №22 – Создание простого диалога списка

В этом уроке мы переходим к разбору третьего стиля диалога: **DIALOG_STYLE_LIST**. Давайте рассмотрим данный стиль диалога на следующем примере. Допустим нам нужно командой, вызвать меню со списком оружия на выбор.

Создаем простейшую команду и внутрь ее пишем функцию вызова диалога. Так как это будет список, нам нужно использовать **\n** – переход на новую строчку. Т.е. мы пишем название пункта \n название пункта и снова \n.

```
1  if (strcmp("/weaponlist", cmdtext, true, 10) == 0)
2  {
3  ShowPlayerDialog(playerid,0,DSL,"Список оружия»,«Кольт\nКольт с глушителем\nПустынный орел", "Выбор", "Отмена");
   return 1;
4  }
5
```

Диалог будет выглядеть следующим образом:



Перед тем как написать функцию для этого диалогового окна, давайте посмотрим на следующую конструкцию приведенную ниже.

```
1  public OnDialogResponse(playerid, dialogid, response, listitem, inputtext[])
2  {
3      if(dialogid == 0)
4          {
5          if(listitem == 1) //Если был выбран 1 пункт списка
6          {
7          //Выполнить этот код
8          }
9          if(listitem == 2) //Если был выбран 2 пункт списка
10         {
11         //Выполнить этот код
12         }
13     }
14      return 1;
15 }
```

Вот так должен выглядеть код для данного стиля диалога. Тут мы проверяем значение listitem, т.е. пункта списка которого мы выберем в диалоге. listitem = 1 – это кольт, значит, в условную конструкцию listitem == 1 мы должны вписать функцию **GivePlayerWeapon**, чтобы дать оружие игроку.

```
if(listitem == 1) //Если был выбран 1 пункт списка
{
	GivePlayerWeapon(playerid,22,100); //Дать игроку кольт и 100 патронов
}
```

То же самое делаем и для остальных **listitem**. Вписываем ту же строчку, только вместо 22, пишем следующее: 23 – для кольта с глушителем, 24 – для пустынного орла (эти цифры, это ID модели этого оружия). ID модели смотрите на странице «ID оружия».

Вообще, приведенный выше пример, очень схож с оператором **switch**, т.к. мы проверяем значение одного аргумента и в зависимости от этого значения выполняем действие. Тоесть, это все можно заменить на оператор **switch**, таким образом:

```
public OnDialogResponse(playerid, dialogid, response, listitem, inputtext[])
{
    if(dialogid == 0)
    {
        switch(listitem)
        {
            case 1: //Если был выбран 1 пункт списка
            {
                //Выполнить этот код
            }
            case 2: //Если был выбран 1 пункт списка
            {
                //Выполнить этот код
            }
        }
    }
     return 1;
}
```

Точно также можно поступить и с параметром dialogid, кому как удобнее.

Урок №23 – Расстановка объектов в Ingame Object Editor

В этом уроке я научу вас создавать простые двери, открываемые по команде. Таким же образом вы сможете потом ставить ворота и шлагбаумы. И начну я с фильтр скрипта **Ingame Object Editor**. (данный скрипт прилагается к учебнику). Почему именно он, когда предлагают много раз MTA Map Editor. Во-первых, это как один из простых способов расставить небольшое количество объектов, если ставить много объектов, на это уйдет больше времени, чем в MTA Map Editor. Во-вторых, это тоже было бы полезно знать. В-третьих, не нужно конвертировать код из map в pwn, все объекты будут в вашем моде вместе с этим фильтр скриптом. Доступ к его опциям имеет только RCON администратор, поэтому вам необходимо будет войти на сервер как RCON администратор.

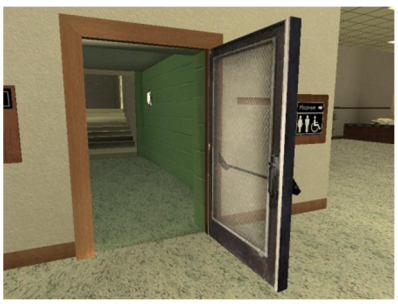
Начнем с создания двери. Войдите на сервер как RCON администратор. Итак, вот место куда мы, например, хотим поставить дверь.

Мы можем поставить деревянную дверь с ID = 1491, это дверь, которая открывается, когда ее толкаешь, но нам нет смысла сейчас ее ставить, потому что мы делаем закрытую дверь, которую нужно будет открыть по команде. Поставим дверь, например с ID = 1500. Итак, в чате пишем следующую команду **/oadd 1500 door**. Команда **/oadd** – добавляет новый объект, через пробел пишется ID объекта и далее имя объекта, любое.

Итак, мы создали нашу дверь, но она оказалась в воздухе. Давайте ее опустим. Пишем следующую команду в чат: **/omode m_z** – эта команда позволит двигать дверь в двух направлениях (вверх или вниз). Дальше садимся на кнопку C и камера переключается на сам объект. Стрелками двигаем дверь вниз к полу. Итак, мы опустили дверь, мы можем снова нажать C, и встать в удобное вам место для обзора объекта и снова сесть. Дальше нам нужно повернуть дверь левее (исходя из рисунка). Пишем туже команду, но уже не m_z а m_xy.

Это команда позволит двигать объект во все четыре стороны (вперед, назад, влево и вправо). Таким образом, двигаем дверь в дверной проем. Нам нужно записать куда-нибудь координаты этой двери в положении закрыто. Координаты этой двери находится в scriptfiles/oed/BREAD_OED.txt и выглядеть они будут следующим образом (чтобы не запутаться, после координат написано имя объекта, имеющего эти координаты, его мы не трогаем, но при вставке в скрипт имя необходимо удалить вместе с лишней запятой):

```
1 1500,371.234283,166.662429,1007.383850,0.000000,0.000000,0.000000,1
```



Итак, мы записали координаты в безопасное место. А теперь повернем дверь в положение открыто. Чтобы повернуть дверь, мы все также используем команду **/o_mode**, только теперь вместо m_xy пишем r_z. И поворачиваем дверь в такое положение:

Теперь снова записываем те же координаты двери в положении закрыто. В координатах ничего не изменилось, кроме предпоследней цифры.

```
1 1500,371.234283,166.662429,1007.383850,0.000000,0.000000,93.000000,1
```

/o_mode r_z – позволяет нам вращать объект таким образом, чтобы дверь открывалась и закрывалась. Есть также r_xy – с помощью, которой уже вращаем дверь в другом направлении.

Запомните m_z и m_xy – это перемещение по осям: z и xy, а r_z и r_xy – это вращение по тем же осям.

Итак, мы получили координаты закрытой и открытой двери. Теперь мы можем удалить этот объект, т.к. у нас есть его координаты. Удаляем его командой **/odel [имя объекта]** в данном случае имя объекта – door, значит, пишем **/odel door**. Удаляем мы его для того, чтобы у нас не получилось 2 копии двери, т.к. одну дверь мы вставим в мод.

Прежде чем приступить к делу объясню ниже приведенные цифры. Первая цифра означает ID модели, следующие 3 цифры это координаты положения объекта, далее идут 3 цифры это координаты вращения объекта, последняя цифра это видимость объекта.

```
1500,371.234283,166.662429,1007.383850,0.000000,0.000000,93.000000,1
```

Итак, приступим к написанию команды. Сначала в начале скрипта мы объявляем глобальную переменную, например door. Затем внутри автовызываемой функции OnGameModeInit – мы присваиваем этой переменной, созданный объект.

```
cpd = CreateObject(1500,371.234283,166.662429,1007.383850,0.000000,0.000000,0.000000,300.0);
```

Видимость мы ставим 300.0, иначе вы будете видеть дверь, только подойдя к ней вплотную. Затем пишем две команды, одну для открытия двери, другую для закрытия:

```
if(strcmp("/open", cmdtext, true, 10) == 0)
{
    SetObjectRot(cpd,0.000000,0.000000,93.000000);
    return 1;
}
if(strcmp("/close", cmdtext, true, 10) == 0)
{
    SetObjectRot(cpd,0.000000,0.000000,0.000000);
    return 1;
}
```

В функции мы указывает координаты вращения, т.к. мы будем вращать объект.

Недостаток этого скрипта только в том, что дверь открывается мгновенно. Все из-за того, что в функции **SetObjectRot** которая, устанавливает координаты вращения объекта, нет последнего аргумента speed, который есть в функции MoveObject.

MoveObject – передвигает объекты в указанное место и так как у него есть аргумент speed, передвигает он объекты плавно. На его основе делают выдвижные двери.

Так вот, давайте сделаем раздвижные двери. Естественно нашу дверь нужно подвинуть немного вперед в дверной проем, чтобы, когда дверь открывалась, ручки двери не торчали из стены. Т.е мы заново получаем координаты открытой и закрытой двери. И теперь мы из полученных координат берем координаты положения, а не координаты вращения.

```
if(strcmp("/open", cmdtext, true, 10) == 0)
{
    MoveObject (cpd,372.531341,166.374206,1007.370971,2.0);
    return 1;
}
if(strcmp("/close", cmdtext, true, 10) == 0)
{
    MoveObject(cpd,371.238067,166.374206,1007.370971,2.0);
    return 1;
}
```

Урок №24 – Запись в файл

В этом уроке я научу вас записывать в файл данные стандартными средствами Pawn, т.е. без использования посторонних инклудов. Итак, приступим.

Сначала мы рассмотрим запись в файл. Нам нужно вручную создать новый файл в папке scriptfiles. Создадим файл Nicks.txt. Дальше переходим к написанию скрипта, создаем переменную для хранения имени игрока и дальше открываем файл, как это показано в скрипте ниже:

```
1  new plname[24];
2  File:nFile = fopen("Nicks.txt",io_append); //Открывает файл для добавления в него записи
```

nFile – это как идентификатор файла, название его может быть любое, я выбрал nFile.
fopen – открывает файл.
io_append – это метод открытия файла, все методы приведены в таблице ниже:

io_write	Записывает в файл, очищает весь ранее записанный текст
io_read	Читает файл, файл должен существовать или крах сервера
io_append	Добавление в файл записи
io_readwrite	Читает файл или создает новый

Давайте сделаем так, чтобы при подключении игрока, его имя записывалось в файл.

```
1  public OnPlayerConnect(playerid)
2  {
3      new plname[24];
4      GetPlayerName(playerid,plname,MAX_PLAYER_NAME); //Получаем имя и записываем в plname
5      File:nFile = fopen("Nicks.txt",io_append); //Открывает файл для добавления в него записи
6      fwrite(nFile, plname); //Записываем имя игрока в файл
7      fclose(nFile); //Закрываем файл
8  }
```

Но у скрипта есть большой недостаток, он записывает имена в строчку, что весьма неудобно. Давайте сделаем так, чтобы имена игроков были в столбик. Для этого мы создадим переменную для хранения отформатированной строки (string).

```
1  public OnPlayerConnect(playerid)
2  {
3      new plname[24], string[24];
4      GetPlayerName(playerid,plname,MAX_PLAYER_NAME);
5      format(string,sizeof(string),"%s\r\n",plname);
6      File:nFile = fopen("Nicks.txt",io_append); //Открывает файл для добавления в него записи
7      fwrite(nFile, string); //Записываем имя игрока в файл
8      fclose(nFile); //Закрываем файл
9  }
```

\n – начинает новую строку
\r – убеждается, что строка начинается сначала, а не где-нибудь посередине.
После записи в файл, его обязательно нужно закрывать.

Урок №25 - Чтение и запись файлов с помощью mxINI

В этом уроке я расскажу, как научится работать с include – mxINI. На момент написания этого урока последняя актуальная версия инклуда 0.5. Прежде чем приступить к изучению инклуда, рекомендую его скачать. На основе этого инклуда в основном делают регистрацию на сервере. В этом уроке я не буду показывать вам, как написать простейшую регистрацию на сервере, а всего лишь объясню, как с помощью mxINI происходит запись и чтение из файла. Думаю, выяснив, как работает данный инклуд, вы сами сможете написать простую регистрацию, возможно даже на диалогах, так как урок с диалогами вы уже проходили.

Файл mxINI.inc нужно скопировать в папку include в директории с редактором Pawno. В начале вашего скрипта обязательно нужно объявить этот инклуд, также как объявляется инклуд a_samp.

Итак, приступим к разбору основных функций mxINI:

```
1  ini_createFile(“filename.ini”); //Создает файл с именем filename.ini в папке scriptfiles.
2  ini_openFile(“filename.ini”); //Открывает файл
3  ini_closeFile(ID открытого файла); //Закрывает файл
```

Запись данных в файл:

```
1  ini_setString(ID файла, "имя ключа", "текст"); // Эта функция записывает текст в ключ.
2  ini_setInteger(ID файла, "имя ключа", 123456 ); // Эта функция записывает целое число в ключ.
3  ini_setFloat(ID файла, "имя ключа", 3.1416 ); // Эта функция записывает десятичное число в ключ.
```

Чтение данных из файла

```
1  ini_getString(ID файла, "имя ключа", <имя переменной>); // Эта функция записывает текст в переменную из
   файла.
2  ini_getInteger(ID файла, "имя ключа", <имя переменной>); // Эта функция записывает целое число в
   переменную из файла.
3  ini_getFloat(ID файла, "имя ключа", <имя переменной>); // Эта функция записывает десятичное число в
   переменную из файла.
```

Запись в файл

Давайте на простом примере с помощью команды, запишем данные игрока: имя, деньги и количество здоровья. Сначала мы создаем 2 переменные, где у нас будут храниться имя игрока и отформатированное сообщение.

```
1  new string[255], PlayerName[32];
2  GetPlayerName(playerid,PlayerName,32);
3  format(string,64,”%s.ini”,PlayerName);
4  new iniFile = ini_createFile(string);
```

После переменных мы узнаем имя игрока функцией **GetPlayerName**. Дальше форматируем строку, как будет называться файл, в который будет производиться запись. В данном случае файл будет называться по имени игрока. Дальше создаем переменную **IniFile**, которая будет хранить handle файла, по которому мы будем обращаться к файлу. Handle файла возвращают функции открытия и создания файла, как показано на 4 (строке) на примере функции создания файла.

Перед тем как записать количество денег и здоровья игрока, нужно сначала их узнать:

```
new money = GetPlayerMoney(playerid);
new Float:health = GetPlayerHealth(playerid);
```

Теперь пишем проверку, если файл выдаст ошибку, открыть файл снова.

```
if(iniFile < 0)
iniFile = ini_openFile(string);
```

Дальше записываем данные в файл и закрываем его:

```
ini_setString(iniFile, "Name", PlayerName); //Записываем имя игрока
ini_setInteger(iniFile, "Money", money); //Записываем количество денег
ini_setFloat(iniFile, "Health", health); //Записываем количество здоровья
ini_closeFile(iniFile); //Закрываем файл
```

Вот так производится запись данных в файл. Запись в файле будет выглядеть примерно следующим образом:

```
Name = Player
Money = 1000
Health = 100.0
```

Чтение из файла

Чтение из файла ничем не сложнее записи, все аналогично. Только тут не нужно проверок, условий: if(iniFile < 0), и в переменной iniFile мы не создаем, а уже открываем созданный нами файл. Весь код будет выглядеть следующим образом:

```
new string[255], PlayerName[32];
new Float:health, money;
GetPlayerName(playerid,PlayerName,32);
format(string,64,"%s.ini",PlayerName);
new iniFile = ini_openFile(string); //Открываем файл
ini_getString(iniFile, "Name", PlayerName); //Узнаем имя игрока и записываем его в переменную
ini_getInteger(iniFile, "Money", money); // Узнаем количество денег и записываем в переменную
ini_getFloat(iniFile, "Health", health); // Узнаем количество здоровья и записываем в переменную
ini_closeFile(iniFile); //Закрываем файл
GivePlayerMoney(playerid,money);
SetPlayerHealth(playerid,health);
```

Урок №26 – Оптимизация функции GetPlayerName

Во многих скриптах, очень часто используется переменная для хранения имени игрока. Она очень много раз создается в различных частях кода. Но можно избежать создания такого большого количества переменных и обойтись одной. Так как в этом случае очень часто используется функция **GetPlayerName**, логично создать отдельную функцию, которая будет вызываться в разных частях кода. Поскольку return может возвращать только целое число или строку, можно написать вот такую простейшую функцию.

```
1  stock PlayerName(playerid)
2  {
3      new PlayerName[MAX_PLAYER_NAME];
4      GetPlayerName(playerid,PlayerName,sizeof(PlayerName));
5      return PlayerName;
6  }
```

Теперь при необходимости узнать имя игрока в определенной части кода, нужно просто вызвать данную функцию. Вот так бы выглядел код с выделением переменной под имя игрока.

```
1  new string[60], plname[MAX_PLAYER_NAME];
2  GetPlayerName(playerid,plname,sizeof(plname));
3  format(string,sizeof(string),"Добро пожаловать к нам на сервер: %s", plname);
4  SendClientMessage(playerid,COLOR_YELLOW,string);
```

А вот так выглядит код с использованием функции, правда стало лучше?

```
1  new string[60];
2  format(string,sizeof(string),"Добро пожаловать к нам на сервер: %s",PlayerName(playerid));
3  SendClientMessage(playerid,COLOR_YELLOW,string);
```

Непонятным остается только одно. Почему разработчики SAMP не решили переделать функцию **GetPlayerName** подобным образом.

Урок №27 – Проверка расстояния между игроками

В этом уроке я научу вас, как делается проверка на расстояние между игроками. Допустим, у нас есть команда передачи денег и нам нужно, чтобы игроки были близко друг к другу, чтобы они могли передавать деньги между собой. В этом уроке я покажу, как делается всего лишь проверка. Итак, приступаем:

Для организации проверки в начало скрипта мы добавляем следующую строку, которая объявит новую автовызываемую функцию, которая будет отвечать за проверку на расстояние между игроками.

```
forward ProxDetectorS(Float:radi, playerid, targetid);
```

Теперь в любом месте вашего скрипта добавляем эту функцию.

```
public ProxDetectorS(Float:radi, playerid, targetid)
{
	if(IsPlayerConnected(playerid)&&IsPlayerConnected(targetid))
	{
		new Float:posx, Float:posy, Float:posz;
		new Float:oldposx, Float:oldposy, Float:oldposz;
		new Float:tempposx, Float:tempposy, Float:tempposz;
		GetPlayerPos(playerid, oldposx, oldposy, oldposz);
		GetPlayerPos(targetid, posx, posy, posz);
		tempposx = (oldposx -posx);
		tempposy = (oldposy -posy);
		tempposz = (oldposz -posz);
		if (((tempposx < radi) && (tempposx > -radi)) && ((tempposy < radi) && (tempposy > -radi)) && ((tempposz <
radi) && (tempposz > -radi)))
		{
			return 1;
		}
	}
	return 0;
}
```

Ну, думаю из кода понятно, что сначала мы проверяем, подключены ли к серверу оба игроками. Затем мы получаем координаты обоих игроков функцией GetPlayerPos. Далее мы вычитаем X-координату игрока передающего деньги, от X-координаты получающего деньги и записываем результат в переменную tempposx, то же самое делаем и с остальными координатами.

Ну а дальше простая математическая проверка, думаю, вам там будет все понятно. Теперь, когда у вас уже готов этот скрипт, можно ставить такие условия:

```
if (ProxDetectorS(5.0, playerid, giveplayerid))
{
	//Тут должен быть код передачи денег или другой код
}
else
{
	SendClientMessage(playerid, COLOR_RED, "Вы находитесь далеко от цели");
}
```

Также в **OnPlayerCommandText** в начале должна быть объявлена переменная **giveplayerid**, которая используется в этой проверке. На этом все, тут довольно все просто если вы уже хорошо знаете условные конструкции, то проблем это не составит.

Урок №28 – Построение сложных команд

Итак, в этом уроке я научу вас писать сложные команды, которые требуют ввода дополнительных параметров после пробела, вы уже знакомы с такими. Вообще даже я сам еще ни разу не писал такого типа команды, однако мне уже ясен принцип работы и написание таких команд. Поэтому я объясню вам, как смогу, а в дальнейшем урок буду корректировать. Я познакомлю вас с функцией **strtok** - это функция, используется, чтобы определить пробел между командой и параметром.

Для этого урока мы рассмотрим команду передачи денег. Для начала нужно, чтобы в начале автовызываемой функции **OnPlayerCommandText** были объявлены все перечисленные переменные:

```
1  new string[255];
2  new cmd[256];
3  new playermoney;
4  new sendername[MAX_PLAYER_NAME];
5  new giveplayer[MAX_PLAYER_NAME];
6  new giveplayerid, moneys, idx;
7  cmd = strtok(cmdtext, idx);
```

Переменная **string** у нас будет хранить текст сообщения о передачи денег между игроками, **cmd** будет хранить весь текст команды. **Sendername** будет хранить имя игрока передающего деньги, **giveplayer** – имя получающего деньги. Переменная **giveplayerid** будет хранить ID игрока получающего деньги, money – будет хранить количество передаваемых денег, а **idx** будет хранить параметры команды. Теперь собственно приступаем к разбору самой команды. После переменных в переменной **cmd** определяем пробел между командой и параметром (7 строка). Все начинается с простейшей команды:

```
1  if(strcmp(cmd, "/givemoney", true) == 0)
2  {
3      return 0;
4  }
```

Внутрь этой команды пишем следующее:

```
1  new tmp[256];
2  tmp = strtok(cmdtext, idx);
3  if(!strlen(tmp))
4  {
5      SendClientMessage(playerid, COLOR_WHITE, "Введите: /givemoney [id игрока] [сумма]");
6      return 1;
7  }
8  giveplayerid = strval(tmp);
9  tmp = strtok(cmdtext, idx);
10 if(!strlen(tmp))
11 {
12     SendClientMessage(playerid, COLOR_WHITE, " Введите: /givemoney [id игрока] [сумма]");
13     return 1;
14 }
15 moneys = strval(tmp);
```

Объясняю. Создаем переменную **tmp**, которая будет хранить параметры команды, это текст, который будет после пробела. Определяем функцией **strtok**, проблем между командой и

параметром (2 строчка). Ставим условие, если параметр не введен, отправляем в чат сообщение. Присваиваем переменной **giveplayerid**, значение первого параметра, которой мы ввели (8 строчка). Снова определяем пробел, но уже между двумя параметрами. Поскольку в команде вводится 2 параметра: ID игрока и сумма денег. Также проверяем, что параметр введен и присваиваем переменной money, значение 2 параметра, которое мы ввели в команде.

Вот пример: ввели вы команду: **/givemoney 1 400**. Переменная **giveplayerid** получает значение 1 параметра, то есть будет равно 1, переменная money получает значение 2 параметра и будет равна 400. Итак, идем дальше:

Дальше мы пишем проверку, подключен ли игрок, получающий деньги к серверу.

```
1  if (IsPlayerConnected(giveplayerid))
2  {
3      GetPlayerName(giveplayerid, giveplayer, sizeof(giveplayer));
4      GetPlayerName(playerid, sendername, sizeof(sendername));
5      playermoney = GetPlayerMoney(playerid);
```

Внутри проверки функцией **GetPlayerName** мы получаем имена обоих игроков. Переменной **playermoney** мы присваиваем количество денег игрока передающего деньги. Итак, мы знаем текущее количество денег игрока и количество денег, которое игрок хочет передать, теперь нам нужно поставить условие, что игрок передает сумму денег не больше той суммы, которая у него с собой есть, пишем такое условие:

```
1  if (moneys > 0 && playermoney >= moneys)
2  {
3      GivePlayerMoney(playerid, (0 - moneys));
4      GivePlayerMoney(giveplayerid, moneys);
5      format(string, sizeof(string), "Вы передали %s (id: %d), $%d.", giveplayer,giveplayerid, moneys);
6      SendClientMessage(playerid, COLOR_YELLOW, string);
7      format(string, sizeof(string), "Вы получили $%d от %s (id: %d).", moneys, sendername, playerid);
8      SendClientMessage(giveplayerid, COLOR_YELLOW, string);
9  }
10 else
11 {
12     SendClientMessage(playerid, COLOR_YELLOW, "Вы не можете передать столько денег.");
13 }
```

Читаю условие: если количество денег, которые игрок хочет передать другому игроку (**moneys**) больше 0 и количество денег, которое у него есть (**playermoney**) больше или равно тому количеству, которое он хочет передать (**moneys**), то мы передаем деньги игроку функцией **GivePlayerMoney**. Отнимаем у игрока передающего количество передаваемых денег, и даем их игроку получающему деньги.

Ну и не забудьте для проверки на подключение игрока получающего деньги поставить else, действие, если игрок не подключен к серверу.

```
1  }
2  else
3  {
4      format(string, sizeof(string), "%d не подключен к серверу.", giveplayerid);
5      SendClientMessage(playerid, COLOR_YELLOW, string);
6  }
```

Ну и напоследок. Если все же при компиляции скрипта pawno будет ругаться на strtok, то добавьте в скрипт эту функцию:

```
strtok(const string[], &index)
{
	new length = strlen(string);
	while ((index < length) && (string[index] <= ' '))
	{
		index++;
	}

	new offset = index;
	new result[20];
	while ((index < length) && (string[index] > ' ') && ((index - offset) < (sizeof(result) - 1)))
	{
		result[index - offset] = string[index];
		index++;
	}
	result[index - offset] = EOS;
	return result;
}
```

Лично мне это функция не понадобилась, сервер я качаю с официального сайта sa-mp.com и там видно эта функция вместе с редактором **pawno** есть.

Урок №29 – Использование return

Return всегда возвращает чего-либо. В данном уроке я научу вас использовать его. При построении сложных команд, будет очень важно знать этот урок. Допустим, у нас есть одно условие внутри команды из функции **OnPlayerCommandText**:

```
1  if(pLogged[playerid] == false)
2  {
3      SendClientMessage(playerid, COLOR_RED, "Вы не авторизованы на сервере");
4  }
5  else
6  {
7      //Выполнить код команды
8  }
```

С помощью return мы можем упростить это условие, отбросив оператор **else**. Для это убираем оператор else и фигурные скобки, выравниваем код команды и получаем следующее:

```
1  if(pLogged[playerid] == false)
2  {
3      SendClientMessage(playerid, COLOR_RED, "Вы не авторизованы на сервере");
4      return 1;
5  }
6  //Выполнить код команды
```

Если условие выполнится функция прекратит дальнейшее свое выполнение, то есть все что после нее будет проигнорировано и функция возвратит **true**. Также return может возврать какое-либо значение. Взять например вот такую функцию, которую вы видели на одном из уроков.

```
1  stock GetName(playerid)
2  {
3      new nick[MAX_PLAYER_NAME];
4      GetPlayerName(playerid, nick, sizeof(nick));
5      return nick;
6  }
```

Функция возвращает имя игрока, то есть строку содержащее имя игрока. Раз функция возвращает имя, можно подставлять функцию в любую часть кода где нужно узнать имя игрока, а не переменную вместо нее, что является оптимизированным способом, чем заводиться для каждой функции переменную.

Урок №30 – Таймер

Таймер – вызывает определённый функцию (**public**) в определённое время. Функция может вызваться единожды, либо повторяться через заданные промежутки времени. Для того, чтобы закрепить таймер на определенной функцией (**public**), создаётся такая функция:

```
1  SetTimer("NewFunction",1500,1);
```

Таймер показан на примере **public** NewFunction. Давайте его вызовем в функции **OnGameModeInit**. Нам нужно объявить функцию имя которой указано в таймере.

```
1  forward NefFunction();
```

Вот код простейший код этой функции.

```
1  public NewFunction(playerid)
2  {
3      print("Hello, World");
4      return 1;
5  }
```

В результате всего вышеуказанного, запустив сервер, в консоли каждые 1,5 (полторы) секунды будет выводиться текст «Hello, World».

Урок №31 – Таймер с передачей параметров

Есть еще один вид таймера, который выполняет функцию, в которую необходимо также передать параметр. Перед тем как мы рассмотрим пример давайте ознакомимся с типами параметров передаваемых данным видом таймера.

Параметр	Тип параметра
b	Вставка числа в двоичной системе счисления.
c	Вставка одного символа.
d	Вставка (целого) числа.
f	Вставка десятичного числа.
i	Вставка числа (integer).
s	Вставка строки.
x	Вставка числа в шестнадцатеричной системе счисления.
%	Вставка символа '%'

Давайте рассмотрим следующий пример.

```
1  SetTimerEx("Say",1000,1,"s","Hello,World");
```

У нас есть таймер вызывающий функцию NewFunction ежесекундно с параметров передающим целочисленное значение переменной **value**. Создадим эту **public** функцию. Сначала мы ее объявим в начале скрипта:

```
1  forward Say(str[]);
```

Ну а теперь код самой функции

```
1  public Say(str[])
2  {
3      printf("%s",str);
4      return 1;
5  }
```

В итоге мы каждую секунду передаем строку «Hello, World» в функцию Say, которая выполняется ежесекундно и отправляет ежесекундно эту строку в консоль сервера.

Урок №32 – Создание сложных команд с помощью dcmd

DCMD - командный процессор, с помощью которого можно более просто (обходя **strtok**) создавать команды с параметрами после пробела. Функция создается с помощью константы (**#define**). В начале скрипта пишем следующее:

```
#define dcmd(%1,%2,%3) if (!strcmp((%3)[1], #%1, true, (%2)) && ((((%3)[(%2) + 1] == '\0') && (dcmd_%1(playerid, ""))) || (((%3)[(%2) + 1] == ' ') && (dcmd_%1(playerid, (%3)[(%2) + 2])))))) return 1
```

Дальше в начале функции **OnPlayerCommandText** объявляем новую команду. Возьмем к примеру команду, которая будет восстанавливать здоровье игрока.

```
public OnPlayerCommandText(playerid,cmdtext[])
{
   dcmd(healplayer,10,cmdtext);
   return 1;
}
```

healplayer - это наша команда. Дальше мы пишем саму команду, но уже в любой части скрипта, как будто как новый **public.**

```
dcmd_healplayer(playerid,params[])//Создана команда healplayer
{
   new player1; //создана переменная
   player1 = strval(params[0]);//к player1 присваивается числовое значение params[0]
   if(IsPlayerConnected(player1))//Проверка, подключён ли игрок к серверу
   {
      if(IsPlayerAdmin(playerid))//Проверка, является ли игрок, вводивший команду, rcon-админом
         {
            SetPlayerHealth(player1,100);//ХП того игрока становиться 100
         }
   }
   return 1;
}
```

Урок №33 – Использование split

Этот довольно сложный урок, поэтому я постараюсь объяснить вам его как можно понятнее. Допустим, нам нужно сохранить транспорт в файл. Мы, конечно, можем воспользоваться любыми методами записи, которые мы знаем, и записать координаты возрождения (spawn) транспорта, но они получатся в столбик, что очень неудобно. Ведь три параметра относятся к одному транспорту и удобнее всего записать все параметры в строку, чтобы каждая строка в файле была только для определенного транспорта.

В этом уроке вы научитесь считывать данные с файла в строках, где указано не 1, а несколько параметров сразу разделенных запятой или другим знаком. Функция **split** как раз и отделяет эти параметры.

Первый аргумент функции, это строка с параметрами, параметры которой нужно разделить. Второй аргумент функции это двумерный массив, куда нужно поместить извлеченные параметры. Третий аргумент функции, это строка с символом, который разделяет эти параметры. Если указать неправильный символ, функция не сработает.

Для начала в любом месте скрипта пишем эту функцию:

```
stock split(const strsrc[], strdest[][], delimiter)
{
    new i, li;
    new aNum;
    new len;
    while(i <= strlen(strsrc)){
      if(strsrc[i]==delimiter || i==strlen(strsrc)){
         len = strmid(strdest[aNum], strsrc, li, i, 128);
         strdest[aNum][len] = 0;
         li = i+1;
         aNum++;
           }
           i++;
    }
    return 1;
}
```

Теперь объявим массив **enum** и глобальную переменную:

```
enum zInfo
{
   Float:zX,
   Float:zY,
   Float:zZ,
};
new CarInfo[3][zInfo];
```

Массив **zInfo** будет хранить считанные с файла данные. Внутри массива мы объявляем переменные для хранения координат X,Y,Z (zX, zY, zZ).

Сразу обратите внимание. Чтобы обратиться к переменной из массива **zInfo**, чтобы передать ее значение другой переменной, нам нужно использовать массив **CarInfo**. Первая цифра в квадратных скобках, это количество строк в массиве, во вторых квадратных скобках будет имя ячейке, к которой мы обращаемся, например: **zX**. То есть обращение к ячейке массива будет выглядеть так: CarInfo[5][zX] (мы обращаемся к 5 строке массива, к ячейке **zX**.

Саму загрузку данных из файла нужно делать в отдельном **public**, как показано на примере ниже.

```
1  stock LoadCars()
2  {
3      new arrCoords[3][64];
4      new strFromFile2[256];
5      new File: file = fopen("cars.cfg", io_read);
6      if (file)
7      {
8          new idx;
9          while (idx < sizeof(CarInfo))
10         {
11             fread(file, strFromFile2);
12             split(strFromFile2, arrCoords, '|');
13             CarInfo [idx][ zX] = floatstr(arrCoords[0]);
14             CarInfo [idx][ zY] = floatstr(arrCoords[1]);
15             CarInfo [idx][ zZ] = floatstr(arrCoords[2]);
16             idx++;
17         }
18         fclose(file);
19     }
20     return 1;
21 }
```

Idx – это номер строки. С помощью цикла **while** мы поочередно считывает с каждой строки файла координаты. Сейчас объясню поэтапно процесс считывания.

Создаем новый массив для временного хранения загруженных данных: **arrCords** (строка 3). Первая цифра в квадратных скобках должна соответствовать цифре массива **CarInfo**. Создаем переменную **strFromFile2**, куда будет извлекаться полученная из файла строка с параметрами (строка 4). Дальше мы открываем файл **cars.cfg**, он должен обязательно существовать в директории **scriptfiles**, иначе возможен крах сервера (строка 5). После проверки создается переменная для хранения номеров строк (строка 8). После переменной идет цикл **while** (строки 9-17). В цикле первым делом мы считываем первую строку файла и записываем в переменную: **strFromFile2** (строка 11). Параметры в файле должны быть разделены знаком, | так как показано в скрипте между одинарных кавычек, а иначе данные будут считаны неправильно. Функцией **split** мы отделяем параметры от строки и записываем в массив **arrCords** по разным ячейкам (строка 12). Ну а дальше мы копируем считанные данные в массив **CarInfo**, перед этим обязательно с помощью функции **floatstr**, мы преобразуем строку, которую мы получили в вещественное число, так как координаты это вещественные числа, а не строки (строки 13-15) и мы их передаем в переменные вещественного типа. В конце массива мы переходим на новую строку и повторяем наш цикл до тех пор, пока цикл не считает 3 строки массива.

В общем, надеюсь, на этом примере вы сможете разобраться в этой функции, так как это уже 33 урок, и вы уже многое прошли. На этом урок заканчивается.

Урок №34 – Функция получения скорости игрока или транспорта

В этом уроке я расскажу вам о функции проверки скорости игрока, и как она работает.

Создается переменная, после которой идет условие, если игрок, находится в какой либо машине. Если условие истинно, проверяем скорость, с которой движется транспорт, если условие ложно, проверяем, с какой скоростью движется игрок.

Дальше путем сложных подсчетов определяется скорость.

```
1  GetPlayerSpeed(playerid)
2  {
3     new Float:ST[4];
4     if(IsPlayerInAnyVehicle(playerid))GetVehicleVelocity(GetPlayerVehicleID(playerid),ST[0],ST[1],ST[2]);
5     else GetPlayerVelocity(playerid,ST[0],ST[1],ST[2]);
6     ST[3] = floatsqroot(floatpower(floatabs(ST[0]), 2.0) + floatpower(floatabs(ST[1]), 2.0) +
   floatpower(floatabs(ST[2]), 2.0)) * 150.0;
7     return floatround(ST[3]);
8  }
```

Функция **floatlabs** - возвращает абсолютную величину отрицательного или положительного числа, **floatpower** - возводит вещественное число в степень, **floatsqroot** - вычисляет квадратный корень данного вещественного числа.

Данная функция возвращает скорость игрока либо транспорта (8 строка). Поэтому если вы, например, хотите записать эту скорость в переменную, допустим **Speed**, пишем так:

```
1  new Speed = GetPlayerSpeed(playerid);
```

На этом я закончу данный урок.

Урок №35 – Создание TextDraw

В этой уроке я научу вас создавать TextDraw. Рассмотрим несколько примеров и создадим свой TextDraw. Для создания текстдравов мы воспользуемся скриптом **TextDrawEditor** (прилагается к учебнику). Он взят с форума официально сайта SAMP (sa-mp.com).

Итак, вводим команду **/text** для открытия меню создание текстдрава. Выбираем пункт «New Project», далее вводим имя проекта. Дальше нам предлагается на выбор: экспортировать проект, удалить проект, создать новый текстдрав. Выбираем пункт «Create new Textdraw» и в меню появляется дополнительный пункт «TDraw 0 'New Textdraw'», это и есть наш созданный текстдрав, теперь можно приступить к редактированию данного текстдрава.

Выбрав данный пункт сверху экрана вы увидите текстдрав «New Textdraw». Выбрав пункт «Change text string», вы можете изменить текст текстдрава, вот пример:

Исходный код данного текстдрава указан ниже:

```
1  new Text:Textdraw0;
2  Textdraw0 = TextDrawCreate(1.000000, 1.000000, " avalanche-net.ucoz.ru");
```

Рассмотрим следующий пункт меню, под названием: «Change Position». То есть изменить позицию текстдрава на экране. Начало координат исходит из левого верхнего угла экрана (для сведения). Итак, можно вбить координаты поочередно вручную, с помощью подпункта меню «Write exact position» или же можно двигать текстдрав стрелками клавиатуры с помощью пункта «Move the Texdraw». Итак, выбираем второй пункт и двигаем текстдрав.

Вот пример расположения текстдрава над радаром, координаты полученные данным скриптом, вы вставляется в функцию TextDrawCreate, указанную на примере кода выше.

Движение текстдрава происходит по координатам X и Y. Клавишами влево или вправо движение текстдрава происходит по координате X, чем цифра меньше, тем текстдрав ближе к левому краю экрана. Клавишами вверх и вниз движение текстдрава происходит по координате Y, чем цифра меньше, тем выше текстдрав.

Вот исходный код текстдрава:

```
1  new Text:Textdraw0;
2  Textdraw0 = TextDrawCreate(12.000000, 320.000000, " avalanche-net.ucoz.ru");
```

Итак, следующий пункт меню: «Change alignment» - Изменить выравнивание. Соответственно без объяснений: по левому краю, по центру и по правому краю. Отложим пример использования данного пункта и приступим к следующему пункту меню: «Change color text».

Тут у нас выбор из трех пунктов:

Write an hexandecimal number – вам предложат ввести HEX код цвета, это тот код, который обычно используется в константах типа COLOR_RED, COLOR_GREEN и т.д.

Use color combinatory – вам предложат ввести цвет, с помощью цифр от 0 до 255, всего четыре цифры, по системе RGB.

Select a premade color – вам предложат выбрать цвет из заранее подготовленных цветов.

Следующий пункт «Change font» позволяет изменить стиль текста. Всего их четыре на выбор. Самый верхний пример, который показан в этом уроке, это 1-ый стиль. Вот все остальные:

Стиль текста - 0

Стиль текста - 2

Стиль текста - 3

Следующий пункт «Change proportionaly» - если включить, то буквы будут более разрозненными. Пункт «Change font size» содержит два подпункта:

Write exact position – позволяет вам указать X и Y координату.

Move the Texdraw – позволяет стрелками клавиатуры визуально изменять размер текстдрава. По координате X – изменяется ширина текстдрава, а по координате Y – изменяется его высота.

Следующий пункт «Edit outline» - редактировать обводку. При включении обводки, текст текстдрава будет обведен таким образом:

При включении такой обводки, использование тени бессмысленно, так как ее не будет видно. Следующий пункт «Shadow Size» изменяет размер тени, а пункт «Shadow/Outline color» изменяет цвет тени или обводки, с помощью тех же 3 способов, что нам предлагались при смене цвета текстдрава.

Следующий пункт «Exit Box» - редактировать рамку. Соответственно при включении рамки, в меню появляются 3 пункта: выключить рамку, изменить размер, изменить цвет. Но это вам уже итак понятно, операции аналогичные вышеуказанным. Ну и естественно два последних пункта: «Дублировать текстдрав» и «Удалить текстдрав».

В общем, у меня получился вот такой текстдрав:

Исходный код, своего готового текстдрава, вы можете найти в файле **text1.tde** в директории сервера scriptfiles. Данный файл нужно открыть любым текстовым редактором, и вырезать в ваш скрипт только нужные строчки. Вот исходный код этого текстдрава:

```
new Text:Textdrawo;
Textdrawo = TextDrawCreate(12.000000, 320.000000, " avalanche-net.ucoz.ru");
TextDrawBackgroundColor(Textdrawo, 255);
TextDrawFont(Textdrawo, 1);
TextDrawLetterSize(Textdrawo, 0.500000, 1.000000);
TextDrawColor(Textdrawo, -1);
TextDrawSetOutline(Textdrawo, 0);
TextDrawSetProportional(Textdrawo, 1);
TextDrawSetShadow(Textdrawo, 1);
TextDrawUseBox(Textdrawo, 1);
TextDrawBoxColor(Textdrawo, 255);
TextDrawTextSize(Textdrawo, 0.000000, 0.000000);
```

Разберем все, что у нас получилось:

TextDrawBackgroundColor – указывает цвет обводки, контура или тени.

TextDrawFont – указывает стиль текста.

TextDrawLetterSize – устанавливает ширину и высоту букв текстдрава.
TextDrawTextSize – устанавливает размер рамки, если она используется.
TextDrawColor – устанавливает цвет текста.
TextDrawAlignment – устанавливает смещение текста.
TextDrawSetOutline – устанавливает размер обводки текста.
TextDrawSetProportional – делает интервал между буквами
TextDrawSetShadow – устанавливает размер тени текста.
TextDrawUseBox – устанавливает рамку позади текстдрава.
TextDrawBoxColor – устанавливает цвет рамки, если она используется.
TextDrawTextSize –

Ну вот и все, пожалуй, на этом закончу данный урок.

Урок №36 – Работа со строками

В этом большом уроке, я покажу вам, как работать со всеми строковыми функциями в строках. Начнем с функции **strlen** – эта функция возвращает длину строки. При подсчете символов, пробелы считаются тоже, так как также являются символами.

```
1  new string[20] = “pawno scripting”;
2  printf(“длина строки: %s”,strlen(string)) //выведет в консоли сервера длину строки 15
```

Далее мы рассмотрим функцию **strval**, которая конвертирует строку в число. У нас есть строка 21. Именно «строка», если через format мы попытаемся вывести через строковой тип «%d», то получим совсем не то, что нам нужно. Если же с помощью данной функции мы сконвертируем строку в число, то мы получим соответствующий результат, вот пример:

```
1  new string[2] = “21”;
2  printf(“значение string = %s”,strval(string)) // 21
```

Есть обратная функция – **valstr**, которая конвертирует число в строку, она аналогична **strval**, только выполняет обратную операцию, поэтому объяснять данную функцию не стану.

Теперь возвратимся к первому примеру, только работать мы будем с функцией **strdel**, которая удаляет часть строки.

```
1  new string[20] = “pawno scripting”;
2  strdel(string,6,15);
3  printf(“%s”, string) //выведет в консоли сервера “pawno”
```

Конечно, если вы хотите вырезать, например с 6 символа до конца строки, то чтобы не считать символы до конца, разумнее вместо цифры 15 поставить функцию **strlen**, которая узнает длину строки. Вот как это показано на примере ниже:

```
1  new string[20] = “pawno scripting”;
2  strdel(string,6, strlen(string));
3  printf(“%s”, string) //выведет в консоли сервера “pawno”
```

А, что если нам нужно извлечь часть символов из строки. В этом нам поможет функция **strmid**.

Допустим, у нас есть две переменные, одна уже содержит текст, а другая пустая. Пишем эту функцию, в скобках пишем имя переменной, в которую будет извлекаться часть строки, дальше через запятую пишем имя переменной, из которой будет извлекаться данная часть строки. А дальше через запятую номер символа, с которого начинается извлекаемая часть и номер символа, с которого извлекаемая часть заканчивается.

```
1  new string[20] = “pawno scripting”;
2  new test[20];
3  strmid(test, string,0, 5);
4  printf(“%s”, test) //выведет в консоли сервера “pawno”
```

Теперь разберем функцию поиска символов в строке – **strfind**. Функция пишется следующим образом, приведу пример без использования переменных, чтобы было виднее.

```
1  new instring = strfind("Are you in here?", "you", true);
```

Если трока найдена, функция возвращает вхождение строки, если нет, она возвращает –1. А следующая функция **strcmp**, известна вам еще при создании простых команд. Она сравнивает строки.

```
1  if(strcmp("/mycommand", cmdtext, true, 10) == 0)
```

Если вам все еще не понятна функция, повторяю еще раз:

strcmp – сравнивает два значения, в данном примере «/mycommand» с командой, которой игрок вводит в чат (**cmdtext**). Значение **true** означает нечувствительность к регистру. Если мы напишем **false**, игрок должен будет соблюдать регистр букв при написании команды.

Ну и последняя функция **strcat** связывает несколько строк в одну. Вот пример для демонстрации работы функции, я думаю, тут также не возникнет вопросов.

```
1  new string[20] = “pawno ”;
2  new test[20] = “scripting”;
3  strcat(string, test);
4  printf(“%s”, string); //выведет в консоли сервера “pawno scripting”
```

Урок №37 – Создание системы логов.

В этом уроке я научу вас создавать систему логов в отдельной функции. Приведу пример команды:

```
if (strcmp("/money", cmdtext, true, 10) == 0)
{
	new pname[24],string[50];
	GivePlayerMoney(playerid,100);
	format(string, sizeof(string), "-- %s получил $100 ", pname);
	SendClientMessageToAll(0xFFFF00AA,string);
	GameLog(playerid,string);
	return 1;
}
```

Так как мы будем создавать новую функцию ведения лога и периодически вызывать ее, в вышеуказанном примере вы уже видите, что я вызываю данную функцию в вышеуказанной команде. Теперь давайте напишем саму функцию.

```
public GameLog(string[])
{
	new entry[144],string[50];
	format(entry, sizeof(entry), "%s\n ", string);
	new File:hFile;
	bFile = fopen( “logs/game.log”, io_append);
	fwrite(hFile, entry);
	fclose(hFile);
}
```

То есть мы передаем сообщение из команды в вышеуказанную функцию и записываем данное сообщение в файл. Теперь я думаю, вам не составит труда сделать такую же функцию под свои нужды. Вот и все, на этом урок закончен!

Урок №38 – Создание входа и выхода из здания.

Итак, чтобы сделать вход в здание, нам нужно проверить, стоит ли игрок перед входом с помощью функции **PlayerToPoint**. Вход мы сделаем по клавише. Давайте сделаем вход по команде, а потом сделаем, выполнение этой команды по клавише.

```
1  if(strcmp(cmdtext, "/enter", true) == 0)
2  {
3      if(PlayerToPoint(1.0,playerid,1481.0232,-1771.8905,18.7958)) //Вход в мэрию ЛС
4      {
5          SetPlayerInterior(playerid,3);
6          SetPlayerPos(playerid,390.5630,173.7925,1008.3828);
7          SetPlayerFacingAngle(playerid,93.2971);
8      }
9  }
```

Функцией **SetPlayerInterior** – мы устанавливаем интерьер, в котором находится интерьер данного здания. Устанавливаем позицию игрока функцией **SetPlayerPos** и его угол функцией **SetPlayerFacingAngle**.

Вход мы сделали, теперь делаем выход. Теперь (между 8 – 9 строкой) пишем условие **else if** следующего содержания:

```
1  else if(PlayerToPoint(1.0,playerid,390.1724,-173.7887,1008. 3828)) //Вход в мэрию ЛС
2  {
3      SetPlayerInterior(playerid,0);
4      SetPlayerPos(playerid,1481.0232,-1771.8905,18.7958);
5      SetPlayerFacingAngle(playerid,3.9151);
6  }
```

Теперь, чтобы сделать вход и выход в здание по клавише, например **Alt**, в автовызываемую функцию **OnPlayerKeyStateChange** пишем следующее:

```
1  if(newkeys == 1024)
2  {
3      OnPlayerCommandText(playerid,"/enter");
4  }
```

Вот и все, вход и выход из здания сделан.

Урок №39 – Воспроизведение звука для игрока

Итак, для воспроизведения для игрока определенного звука, существует функция **PlayerPlaySound**. Если реализовать данную функцию в вашем скрипте, то в последующем вы сможете проигрывать звук для игрока, всего одной строчкой:

```
1  PlayerPlaySound(playerid, 1187, 0.0, 0.0, 0.0);
```

А вот и сама функция:

```
1  public PlaySoundForPlayer(playerid, soundid)
2  {
3          new Float:pX, Float:pY, Float:pZ;
4          GetPlayerPos(playerid, pX, pY, pZ);
5          PlayerPlaySound(playerid, soundid,pX, pY, pZ);
6  }
```

Честно говоря, сам не понимаю смысл данной функции, когда внутри нее уже есть функция **PlayerPlaySound**, которая делает одно и то же. Но, тем не менее, урок на этом заканчивается.

Урок №40 – Функция Random

В этом уроке мы рассмотрим примеры получения случайного числа, двумя различными функциями Random. Первая функция стандартная, вторая – сторонняя разработанная функция. Вы научитесь не только получать просто случайное число, но и случайное число из массива. Начнем со стандартной функции, синтаксис функции таков:

```
1  random(max); //случайное число между 0 и max
```

Вот пример простейшей команды с выводом случайного числа в чат:

```
1  if(strcmp("/random", cmdtext, true, 10) == 0)
2  {
3      new string[64]; // создаем строку, которая выведется на экран.
4      format(string, 64, "Случайное число: %d", random(10));
5      SendClientMessage(playerid, 0xFFFFFF, string); // выводим сообщение пользователю
6      return 1;
7  }
```

Но, что если нам нужно получить случайное число не от нуля, а от единицы до десяти. Тут мы заранее прибавим к случайному числу единицу. То есть строка 4, в вышеуказанном коде выглядела бы следующим образом.

```
1  format(string, 64, "Случайное число: %d", random(10)+1);
```

Но есть и не стандартная функция, которая уже имеет два параметра: минимальное и максимальное число диапазона, из которого будет браться случайное число. Функция выглядит следующим образом:

```
1  stock Random(min, max)
2  {
3          new a = random(max - min) + min; return a;
4  }
```

Теперь четвертая строка выглядела бы следующим образом:

```
1  format(string, 64, "Случайное число: %d", random(1,10);
```

Данную функцию можно использовать в других различных функциях, например в операторе switch, как это показано на примере ниже:

```
1  switch(random(4)) // генерируем число от 0 до 3 (включая)
2  {
3      case 0: SetPlayerPos(playerid, X1,Y1,Z1); // телепортируем игрока в координаты X1,Y1,Z1
4      case 1: SetPlayerPos(playerid, X2,Y2,Z2); // ...
5      case 2: SetPlayerPos(playerid, X3,Y3,Z3); // ...
6      case 3: SetPlayerPos(playerid, X4,Y4,Z4); // аналогично.
7  }
```

Выше перечисленный скрипт больше подходит для создания какой-нибудь азартной игры. И в завершении этого урока рассмотрим пример с использованием массива. Приведу коротко, так как надеюсь, вы уже изучали массивы. Допустим, вы хотите сделать рандомный спавн игроков. Возьму за основу стандартный пример из скрипта LVDM (идущий в комплект с сервером SAMP).

```
1  new Float:gRandomPlayerSpawns[23][3] = {
```

У нас есть вышеуказанный массив. Теперь нам нужно получить случайные координаты из массива. Для этого мы берем случайное число от количества строк массива. То есть берем случайное число от 0 до 22

```
1  new rand = random(sizeof(gRandomPlayerSpawns));
2  SetPlayerPos(playerid, gRandomPlayerSpawns[rand][0], gRandomPlayerSpawns[rand][1],
   gRandomPlayerSpawns[rand][2]); // Warp the player
```

В вышеуказанном примере мы получаем случайное число строки массива и записываем его в переменную rand, которую потом используем при телепортации игрока к месту спавна.

Вот и все!

Урок №41 – Расстановка объектов в MTA

В этом уроке я расскажу, как расставить объекты для Samp в MTA Map Editor. Я надеюсь, вы уже скачали и установили MTA, так как к учебнику он не прилагается. А теперь приступаем к уроку, запускаем MTA, далее Map Editor.

Сначала кратко об управлении. Для перемещения камеры по штату соответственно используются буквенные стрелки: W A S D. Чтобы ускорить движения камеры во время перемещения, нужно зажать клавишу Shift.

Итак, допустим, вы уже добрались до места, где хотите расстановить объекты. Я выбрал место, например автошколу:

Чтобы скрыть все панели, нажмите клавишу F4. Для того чтобы протестировать ваше творение, нажмите клавишу F5.

Нажимаем клавишу F для активации верхней и нижней панели. С данными панелями вы легко и сами разберетесь.

Нажимаем кнопку **Object** в нижней панели. Появляется большой список объектов на выбор. Чтоб долго не искать нужный объект выберите подходящую категорию объектов из ниспадающего меню сверху, над списком объектов.

Для моей автошколы потребуется, например, для начала небольшие железные заборы. В моем случае, я выбираю категорию structures и подкатегорию Fences, Walls, Gates and Barriers. (Заборы, стены, ворота и барьеры).

Итак, если вы выбрали себе объект, ставьте в нужное вам место. Если зажать клавишу Ctrl и крутить колесико мыши, объект начнет вращаться по оси Z. Если вы хотите поднять или опустить объект нажмите **PageUp** / **PageDown**. Можно вращать объект, если нажимать те же кнопки при зажатой клавише **Ctrl**.

Если вы случайно поставили объект, криво наведите курсор на объект и нажмите правую кнопку мыши. Пока вы держите объект его можно удалить клавишей Delete или если камера расположена неудобным для вас образом, чтобы поставить объект, нажмите F, чтобы передвигаться по карте вместе с объектом.

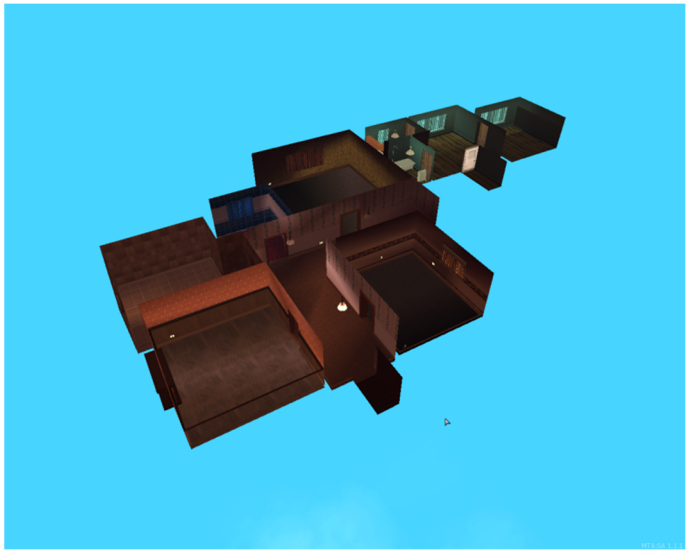
Если вы хотите установить объект, а следом за ним еще один такой же объект, то зажмите **Ctrl** и установите объект. Объект установится, а на вашем курсоре будет закреплена копия этого же объекта, которую можно установить в другое место. То есть вы копируете объект, который держите, для расстановки их в разных местах.

Для более точной установки объекта, поставьте объект, нажмите двойным кликом мыши по нему и появится окно редактирования объекта. Меняйте десятые или сотые доли координат положения объекта X, Y или Z. В этом же окне можно заменить объект на другой объект.

Вот так, я расставил заборы у автошколы. И напоследок если вы хотите расстановить объект в интерьере. В верхнем меню есть такая кнопка «Locations». Нажимаете и выбираете из списка интерьер, и нажимаете кнопку **Go**, там же указаны координаты интерьера. Аналогичным способом теперь вы можете расставлять там объекты.

Урок №42 – Делаем свой интерьер в MTA

В этом уроке, по сути, идет закрепление прошлого 41-го урока. Итак, начнем урок. Нажимаем кнопку Locations в верхней панели и перемещаемся в любой интерьер, рядом с которым мы создадим свой интерьер.



Слева показан созданный мною интерьер, ID объекта: 13701. Все объекты домов в MTA или почти все, находятся в категории: Building и подкатегории Houses and Apartments, приблизительно в конце списка. Справа я показываю созданный интерьер в тестовом режиме, клавиша F5. Теперь можно расставлять мебель в данном интерьере.

Чтобы узнать координаты нового интерьера, установите камеру перед выходом из вашего интерьера, нажмите F и нажмите кнопку Locations. Далее нажмите на кнопку: Dump current position и вы получите необходимые вам координаты.

Урок №43 – Конвертируем map в pwn.

Расставив объекты в MTA, их можно перенести в SAMP при помощи конвертации файла map от MTA в файл SAMP pwn. Конвертировать map в pwn можно как вручную при помощи блокнота, так и при помощи программы.

Файл map вы можете найти в директории C:\Program Files\MTA San Andreas 1.1\server\mods\deathmatch\resources, в которой будет находится папка с названием вашего проекта. В этой папке находится данный файл, с таким же названием, это он и есть.

Можно воспользоваться хорошим онлайн конвертером по адресу: **http://gtamap.delux-host.com/converter/**. Увы я не нашел хорошей программы конвертера данных расширений файлов.

У онлайн активатора минусов нет. Заходите на вышеуказанный сайт, выбираете в ниспадающем меню посередине, пункт Pawn code for SA:MP, загружаете на данный сайт ваш map файл и получаете исходный код pwn.

Вот и все, данный код вы можете вставить в ваш gamemode. На этом я закончу данный урок.

Урок №44 – ReturnUser

В инклуде **utils**, есть одна очень интересная функция. Она помогает нам выполнять команды, такие как, например передача денег, не только по ID игрока, которому мы передадим деньги, но вместо ID также возможно указать имя игрока. То есть функция возвращает ID игрока.

```
1  #include <utils>
```

Подключив данный инклуд к скрипту, вы можете присваивать переменной tmp, значение не через **strval**, как это обычно используется только для передачи ID игрока, а через функцию **ReturnUser**. Вот так, как показано на примере ниже.

```
1  giveplayerid = ReturnUser(tmp);
```

С построением сложных команд вы уже знакомы из 27-го урока и там, в примере вы видели строку вида:

```
1  giveplayerid = strval(tmp);
```

Это значит, команда работает только с ID игрока, но не с его именем. Данную функцию можно использовать не только в командах, но и, например, в диалогах типа DIALOG_STYLE_INPUT. На этом урок закончен!

Урок №45 – Движение объекта

Итак, на основе этого урока вы сможете создавать открывающиеся/закрывающиеся двери или ворота. Для того, чтобы сделать, например дверь, нужно сначала получить координаты положения ворот в состояниях: открыто и закрыто. Допустим, мы уже получили данные координаты. Ниже приведенные координаты получены с помощью скрипта **OngameObjectEditor**.

```
10184,1537.008544,-1451.453491,14.855525,0.000000,0.000000,269.343505 //закрытые ворота
10184,1537.008544,-1451.453491,19.192020,0.000000,0.000000,269.343505 //открытые ворота
```

Итак, делаем команду открытыя и закрытия ворот. Сначала я создал двумерный массив. Первый столбец которого хранит ID объекта, поскольку ворот мы можем делать сколько угодно пока не будет достигнут лимит объектов. Второй столбец массива хранит статус ворот: открыто либо закрыто.

Я заранее получил координаты точки, где будет происходить проверка игрока на местонахождение возле ворот. Я использую функцию **IsPlayerInRangePoint** вместо **PlayerToPoint** так как она стандартная и ее рекомендуется использовать вместо **PlayerToPoint**.

```
if(strcmp(cmdtext, "/gate", true) == 0) //
{
    if(IsPlayerInRangeOfPoint(playerid,7.0,1534.5895,-1450.5264,13.3828))  //ЛС ВОРОТА В КОМ. РАЙОНЕ
    {
        if(Gate[0][1] == 0)
        {
            MoveObject(Gate[3][0],1537.008544,-1451.453491,19.192020,1.0);
            Gate[0][1] = 1;
        }
        else
        {
            MoveObject(Gate[3][0],1537.008544,-1451.453491,14.855525,1.0);
            Gate[0][1] = 0;
        }
    }
}
```

Итак, для передвижения объекта, существует функция **MoveObject**. Ворота изначально у нас закрыты. Gate[0][1] – ссылка на ячейку в массиве хранящую статус ворот (0 – закрыто, 1 – открыто). В данном случае оно у нас равно 0. Перед тем как отрыть ворота нам нужно удостоверить, что игрок стоит возле ворот, и проверить в каком положении в текущий момент находятся ворота. Сначала проверяем положение игрока с помощью функции **IsPlayerInRangePoint** а затем проверяем значение Gate[0][1].

Чтобы открыть ворота, нам нужно в функции MoveObject использовать координаты ворот в положении «открыто». Открыв ворота, мы должны присвоить Gate[0][1] значение 1, то есть изменить статус ворот на «открыто».

Делаем аналогичным образом, но наоборот, чтобы закрыть ворота. Используем координаты ворот в положении «Закрыто» в функции **MoveObject** и присваиваем Gate[0][1] значение 0.

Последним аргументом в функции **MoveObject** не забудьте указать скорость движения объекта. Скорость должна быть десятичным числом.

Урок №46 – Цветной текст в строке

В строку можно передавать не только другие значения, строки или управляющие символы. Передавать можно цвет. Но чтобы передать цвет не обязательно использовать длиные HEX-коды цвета начинающиеся с символов «0x». Вот небольшой список констант с легкими кодами цвета.

```
#define COLOR_GREY "C3C3C3"
#define COLOR_GREEN "6EF83C"
#define COLOR_RED "F81414"
#define COLOR_YELLOW "F3FF02"
#define COLOR_WHITE "FFFFFF"
#define COLOR_ORANGE "FFAF00"
#define COLOR_VIOLET "B700FF"
#define COLOR_LIGHTGREEN "C9FFAB"
#define COLOR _LIGHTBLUE "00C0FF"
#define COLOR _BLUE  "0049FF"
```

Чтобы передать цвет в строку, нужно перед текстом, который вы хотите окрасить в определенный цвет, вставить фигурные скобки, внутри которых должен быть код цвета, который мы в нижеуказанном примере для удобства заменили константой. Обратите внимание, цвет мы передали в строковой параметр функции **SendClientMessage**. Чтобы передать цвет в данную функцию, поскольку в функции уже имеется параметр отвечающий за цвет сообщения, мы должны вторым параметром передать «-1».

```
SendClientMessage(playerid, -1, " { COLOR_GREEN} Зеленый {COLOR_RED} Красный {COLOR_GREY}  Серый");
```

То же самое можно делать в диалогах, других функциях и строках.

Урок №47 – Функции sparam, iparam, fparam

Эти функции служат для вырезания строки из строки разделенные каким либо символом. Каждая функция вырежает только свой тип значения, она так и называется в соответствии со своим типом: sparam – расшифровывается как «string param» - извлекает подстроку под указанным номером, по желанию и вместе с остатком строки. iparam – «integer param» - извлекает подстроку под указанным номером и возвращает ее численное значение и fparam - «float param» - извлекает подстроку под указанным номером и возвращает ее десятичное (Float) значение.

Функция sparam

Синтаксис функции **sparam** следующий:

```
sparam(dest[], maxSize, const source[], delimiter = ' ', substrIndex, withRest)
```

dest[] - строка для вывода результата;
maxSize - максимальная длина подстроки, помещенная в dest[];
source[] - основная строка, из которой вырезаем подстроку;
delimiter - символ, который будет делить основную строку на подстроки;
substrIndex - порядковый номер подстроки;
withRest - логический флаг, если равен 1, то в dest[] будут помещены все подстроки, начинающиеся с указанного номера substrIndex, в том числе и все символы delimiter;
Функция ничего не возвращает!

```
new str[20];
sparam(str, 20, "First|Second|Third", '|', -1);  //str = "", т.к. подстроки с таким номером не существует
sparam(str, 20, "First|Second|Third", '|', 0 );  //str = "First"
sparam(str, 20, "First|Second|Third", '|', 1 );  //str = "Second"
sparam(str, 20, "First|Second|Third", '|', 2 );  //str = "Third"
sparam(str, 20, "First|Second|Third", '|', 1, 1); //str = "Second|Third"
sparam(str, 20, "First|Second|Third", '|', 3); //str = "", т.к. подстроки с таким номером не существует
```

То же самое можно делать в диалогах, других функциях и строках.

Функция iparam

Синтаксис функции **iparam** следующий:

```
iparam (const source, delimiter = ' ', substrIndex)
```

source[] - основная строка, из которой вырезаем подстроку;
delimiter - символ, который будет делить основную строку на подстроки;
substrIndex - порядковый номер подстроки;
Возвращает целочисленное значение подстроки!

```
new value;
value = iparam("First 56 345 8", ' ', -1);  //str = "", т.к. подстроки с таким номером не существует
value = iparam("First 56 345 8", ' ', 0);  //str = 0, т.к "First» это не число.
value = iparam("First 56 345 8", ' ', 1);  //str = "56"
value = iparam("First 56 345 8", ' ', 2);  //str = "345"
value = iparam("First 56 345 8", ' ', 3); //str = "8"
value = iparam("First 56 345 8", ' ', 4); //str = "", т.к. подстроки с таким номером не существует
```

Функция iparam

Синтаксис функции **iparam** следующий:

```
1  fparam(const source, delimiter = ' ', substrIndex)
```

source[] - основная строка, из которой вырезаем подстроку;
delimiter - символ, который будет делить основную строку на подстроки;
substrIndex - порядковый номер подстроки;
Возвращает дробное численное значение подстроки!

```
1  new Float:value;
2  value = fparam("First 56.3 345.26 8", ' ', -1);  //str = "", т.к. подстроки с таким номером не существует
3  value = fparam("First 56.3 345.26 8", ' ', 0);  //str = 0.0, т.к "First» это не число.
4  value = fparam("First 56.3 345.26 8", ' ', 1);  //str = "56.3"
5  value = fparam("First 56.3 345.26 8", ' ', 2);  //str = "345.26"
6  value = fparam("First 56.3 345.26 8", ' ', 3); //str = "8.0"
7  value = fparam("First 56.3 345.26 8", ' ', 4); //str = "", т.к. подстроки с таким номером не существует
```

Сами функции достаточно большие, чтобы демонстрировать их в данном уроке, поэтому есть инклуд под названием **mxParam**, созданный автором этих функций **MX_Master**, по его словам его функции быстрее чем «**strtok**». Данный инклуд прилагается к учебнику.

Урок №48 – Создание 3D надписи.

Если вы хотите создать 3D текст и в дальнейшем его так и оставить, не редактируя отображаемый текст, то его можно просто создать при помощи функции **Create3DTextLabel**. Но если нам нужно в дальнейшем изменить 3D тест, вам нужно заранее создать переменную для хранения идентификатора 3D текста. Данный идентификатор возвращает вышеуказанная функция **Create3DTextLabel**. Но для хранения идентификатора 3D текста нужен специальный тип переменной.

```
1  new Text3D:text;
```

Приставка Text3D: обязательна перед именем переменной, если в ней будет храниться идентификатор 3D текста.

```
1  text = Create3DTextLabel("Ваш текст", COLOR_YELLOW, 1562.5466, 562.4622, 14.7152, 25.0 , 0, 1);
```

Первый аргумент функции **Create3DTextLabel**, это текст который будет отображаться в надписи. Второй аргумент, это цвет текста. Если вы хотите сделать цветной текст, состоящий из разных цветов, установите значение второго аргумента в «-1». Следующие 3 аргумента это координаты местоположения надписи. После десятичное число, это расстояние на котором будет виден 3d текст. Предпоследняя цифра, это ID виртуального мира, в котором будет виден 3D текст. По умолчанию игрок находится в нулевом виртуальном мире. Последний аргумент задает видимости 3D текста сквозь стены: если 0, то будет виден сквозь стены, если 1, то не будет.

Редактирование 3D текста.

Изменить 3D текст вы сможете при помощи функции **Update3DTextLabelText**. Функция позволяет изменить цвет текста и сам текст. Вот так можно изменить 3D текст, который мы создали на вышеуказанном примере:

```
1  Update3DTextLabelText (text, "Новый текст", COLOR_RED);
```

Прикрепление 3D текста к игроку или транспорту

Прикрепление 3D текста к игроку или транспорту происходит по координатам относительно координат игрока. Функция **Attach3DTextLabelToPlayer** осуществляет прикрепление 3D текста к игроку.

```
1  Attach3DTextLabelToPlayer(text,playerid, 0.0, 0.0, 0.7);
```

В последних трех аргументах функции показаны координаты смещения относительно координат игрока. Увеличивая последний аргумент, как это показано на примере, мы поднимаем 3D текст от центра игрока вверх, то есть мы можем таким образом разместить созданный 3D текст над головой у игрока. Уменьшая цифру, то есть, делая ее отрицательной, мы опускаем текст. Также манипулируя Y координатой мы двигаем текст вперед и назад, манипулируя X координатой двигаем текст вправо и влево от текущего положения игрока.

Ну, собственно все также аналогично и с транспортом, функция **Attach3DTextLabelToVehicle** идентична вышеуказанной функции, поэтому вы сами догадаетесь. Ну и напоследок, удалить созданный 3D текст можно функцией **Delete3DTextLabel**, в скобках функции указывается его переменная хранящая идентификатор.

```
1  Delete3DTextLabel(text);
```

Урок №49 – Использование функций gettime и getdate.

Для того чтобы узнать текущее реальное время на сервере существует функция gettime. Следующий пример демонстрирует простую команду для того, чтобы узнать текущее время на сервере. Команда выводит текущее время в чат.

```
1  if(strcmp(cmd, "/time ", true) == 0)
2  {
3    new hour,minute,second;
4    new str[128];
5    gettime(hour,minute,second);
6    format(str,sizeof(str),"Сейчас на сервере %d:%d:%d",hour,minute,second);
7    return SendClientMessage(playerid, 0xFFFFFF,str);
8  }
```

Для того чтобы узнать текущую реальную дату на сервере существует функция getdate. Следующий пример демонстрирует простую команду для того, чтобы узнать текущую дату на сервере. Команда выводит текущую дату в чат.

```
1  if(strcmp(cmd, "/date ", true) == 0)
2  {
3    new day,month,year;
4    new str[128];
5    getdate(day,month,year);
6    format(str,sizeof(str),"Сейчас на сервере %d:%d:%d", day,month,year);
7    return SendClientMessage(playerid, 0xFFFFFF,str);
8  }
```

Сама функция getdate возвращает количество дней прошедших с начала года. В этом можно убедиться на простои примере: добавив нижеприведенную строчку в функцию OnGameModeInit. Результат будет видно в консоли сервера.

```
1  printf("С начала года прошло %d дней",getdate());
```

На момент написания данного урока, дата была 18 января 2011 года, это значит, что в консоли результат должен быть такой: «С начала года прошло 352 дней». Чтобы получить число дней, оставшихся до нового года, нужно от 365 дней отнять значение возвращаемое функцией getdate.

```
1  printf("До нового года осталось %d дней",365-getdate());
```

Соответственно, результат должен быть такой «До нового года осталось 13 дней».

Функция gettime возвращает unix время. К сожалению я с ним не знаком, но скажу, что такое UNIX время.

Unix время

Unix время - это число секунд с 31 декабря 1969 года на 1 января 1970 года, время с этого момента называют «эрой UNIX». Мало кто использует unixtime, но его хорошо использовать для бана на время, вывода сообщений в определённое время вплоть до секунды, и разные другие решения.

Урок №50 – Использование PVar.

Pvar (Per-player variables) - система персональных переменных. Изначально SA-MP был создан для максимум 100 игроков. Это значит, что создание массивов в PAWN размером MAX_PLAYERS, например, PlayerInfo[MAX_PLAYERS], в общем было вполне нормально. Теперь же MAX_PLAYERS равняется 500, а скриптеры создают массивы из 500 элементов лишь для хранения одиночных флагов. Это может оказаться очень затратно в плане потребления памяти. Кроме того, обычные переменные нужно вручную сбрасывать/обнулять, когда игрок покидает сервер.

Преимущества использования PVar'ов над массивами размером в MAX_PLAYERS:

1) PVar'ы доступны из всех загруженных гейммодов и скриптов, упрощая модуляризацию вашего кода.

2) PVar'ы автоматически удаляются, когда игрок выходит с сервера, что означает вам не нужно вручную сбрасывать переменные для следующего подключившегося игрока.

3) Больше нет нужды в сложных enum/playerInfo-структурах.

4) Экономит память, не расходуя ее на элементы под playerid'ы, которые, возможно, вообще никогда не будут использованы.

5) Вы можете легко перечислять и выводить/хранить список PVar'ов. Это упрощает как отладку, так и хранение информации об игроках.

6) Если даже PVar не был создан, при запросе его значения возвращается 0 по умолчанию.

7) PVar'ы могут хранить очень большие строки используя динамически выделяемую память.

PVar`ы могут быть также нескольких типов. Вот небольшое описание функций.

```
1 SetPVarInt(playerid, varname&#91;], int_value); - устанавливает значение для целого PVar`a
2 GetPVarInt(playerid, varname&#91;]); - возвращает значение целого PVar`a
3 SetPVarString(playerid, varname&#91;], string_value&#91;]); - устанавливает значение строкового PVar`a
4 GetPVarString(playerid, varname&#91;], string_return&#91;], len); - копирует значение строкового PVar`a в string_return
5 SetPVarFloat(playerid, varname&#91;], Float:float_value); - устанавливает значение вещественного PVar`a
6 Float:GetPVarFloat(playerid, varname&#91;]); - возвращает значение вещественного PVar`a
7 DeletePVar(playerid, varname&#91;]); - удаляет PVar
```

И соответственно константы типов PVar`ов:

```
1 #define PLAYER_VARTYPE_NONE   0 //неизвестно
2 #define PLAYER_VARTYPE_INT    1 //целое число
3 #define PLAYER_VARTYPE_STRING 2 //строка
4 #define PLAYER_VARTYPE_FLOAT  3 //вещественное число
```

Эти константы нужны для проверок функции **GetPVarType**, которая возвращает тип указанного PVar`a. Ниже приведены функциии

```
1 native GetPVarsUpperIndex(playerid); - узнает максимальный индекс PVar`a для игрока
2 native GetPVarNameAtIndex(playerid, index, ret_varname&#91;], ret_len); - узнает название PVar`a по его индексу
  (внутреннему номеру)
3 native GetPVarType(playerid, varname&#91;]); - возвращает тип указанного PVar`a
```

Итак, рассмотрим пример использование PVar`a на команде. Этой команде мы сделаем ограничение на время перед ее повторным использованием. Сделаем мы этот таким способом, чтобы избежать использование таймера.

Нижеприведенная строка создает инициализированную персональную переменную rtime. Эту строку мы добавляем в тело команды.

```
1 SetPVarInt(playerid,"rtime",gettime()+10); //rtime - переменная которая будет проверятся позже, 10 - кол-во секунд.
```

Над этой строчкой в самое начало команды мы ставим следующую проверку.

```
if(GetPVarInt(playerid,"rtime") > gettime())
{
    SendClientMessage(playerid, -1, "Вы уже использовали эту команду, ждите 10 секунд.");
    return 1;
}
```

Таким образом мы смогли без таймера, организовать задержку перед повторным использованием команды, заодно научились использовать Pvar.

```
if(strcmp(cmdtext, "/command", true) == 0)
{
     if(GetPVarInt(playerid"rtime") > gettime())
     {
        SendClientMessage(playerid, -1, "Вы уже использовали эту команду, ждите 10 секунд.");
        return 1;
     }
     SendClientMessage(playerid, -1, "Команда использована.");
     SetPVarInt(playerid, "rtime",gettime()+10);
     return 1;
}
```

III. Список ресурсов Pawn

В этом разделе приведен ID всего, чего возможно в SAMP.

ID Иконок на радаре

ID		Название
1		Белый Квадрат
2	[AD]	Позиция игрока
3		Игрок (Карта в меню)
4		Север
5		Аэропорт
6		Амуниция
7		Парикмахерская
8		Big Smoke
9		Лодочная станция
10		Закусочная
11		Бульдозер
12		Catalina
13		Cesar
14		Курочка
15		Carl Johnson
16		Crash
17		Кафе
18		Emmet
19		Враждебная Атака
20		Пожарные
21		Подружка
22		Госпиталь
23		Loco
24		Maddog
25		Мафия
26		OG Loc
27		Мастерская

28		*OG Loc*
29		*Well Stacked Pizza Co*
30		*Полиция*
31		*Собственность*
32		*Собственность*
33		*Стадион*
34		*Ryder*
35		*Сохранение игры*
36		*Школа*
37		*Вопрос*
38		*Sweet*
39		*Тату салон*
40		*The Truth*
41		*Метка пути*
42		*Toreno*
43		*Триады*
44		*Казино триад*
45		*Одежда*
46		*Woozie*
47		*Zero*
48		*Дискотека*
49		*Бар*
50		*Ресторан*
51		*Грузовик*
52		*Ограбление*
53		*Гонки*
54		*Спортзал*
55		*Автомобиль*
56		*Свет*
57		*Путь к аэропорту*
58		*Varrios Los Aztecas*
59		*Ballas*
60		*Los Santos Vagos*
61		*San Fierro Rifa*
62		*Grove street*
63		*Покраска*

ID Звуков

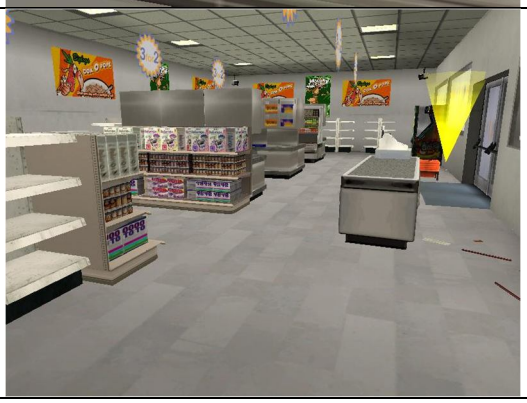
San Andreas Multiplayer 0.3C	
ID	***Название***
1002	*SOUND_CEILING_VENT_LAND*
1009	*SOUND_BONNET_DENT*
1020	*SOUND_CRANE_MOVE_START*
1021	*SOUND_CRANE_MOVE_STOP*
1022	*SOUND_CRANE_EXIT*
1027	*SOUND_WHEEL_OF_FORTUNE_CLACKER*
1035	*SOUND_SHUTTER_DOOR_START*
1036	*SOUND_SHUTTER_DOOR_STOP*
1039	*SOUND_PARACHUTE_OPEN*
1052	*SOUND_AMMUNATION_BUY_WEAPON*
1053	*SOUND_AMMUNATION_BUY_WEAPON_DENIED*
1054	*SOUND_SHOP_BUY*
1055	*SOUND_SHOP_BUY_DENIED*
1056	*SOUND_RACE_321*
1057	*SOUND_RACE_GO*
1058	*SOUND_PART_MISSION_COMPLETE*
1062	*SOUND_GOGO_TRACK_START (музыка)*
1063	*SOUND_GOGO_TRACK_STOP (музыка)*
1068	*SOUND_DUAL_TRACK_START (музыка)*
1069	*SOUND_DUAL_TRACK_STOP (музыка)*
1076	*SOUND_BEE_TRACK_START (музыка)*
1077	*SOUND_BEE_TRACK_STOP (музыка)*
1083	*SOUND_ROULETTE_ADD_CASH*
1084	*SOUND_ROULETTE_REMOVE_CASH*
1085	*SOUND_ROULETTE_NO_CASH*
1095	*SOUND_BIKE_PACKER_CLUNK*
1097	*SOUND_AWARD_TRACK_START (музыка)*
1098	*SOUND_AWARD_TRACK_STOP (музыка)*
1100	*SOUND_MESH_GATE_OPEN_START*
1101	*SOUND_MESH_GATE_OPEN_STOP*
1130	*SOUND_PUNCH_PED*
1131	*SOUND_AMMUNATION_GUN_COLLISION*

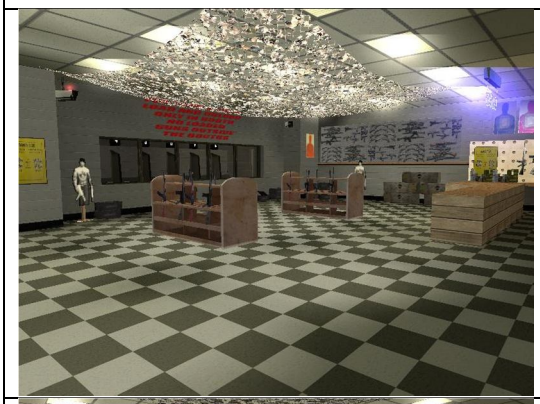
1132	*SOUND_CAMERA_SHOT*
1133	*SOUND_BUY_CAR_MOD*
1134	*SOUND_BUY_CAR_RESPRAY*
1135	*SOUND_BASEBALL_BAT_HIT_PED*
1136	*SOUND_STAMP_PED*
1137	*SOUND_CHECKPOINT_AMBER*
1138	*SOUND_CHECKPOINT_GREEN*
1139	*SOUND_CHECKPOINT_RED*
1140	*SOUND_CAR_SMASH_CAR*
1141	*SOUND_CAR_SMASH_GATE*
1142	*SOUND_OTB_TRACK_START*
1143	*SOUND_OTB_TRACK_STOP*
1144	*SOUND_PED_HIT_WATER_SPLASH*
1145	*SOUND_RESTAURANT_TRAY_COLLISION*
1147	*SOUND_SWEETS_HORN*
1148	*SOUND_MAGNET_VEHICLE_COLLISION*
1149	*SOUND_PROPERTY_PURCHASED*
1150	*SOUND_PICKUP_STANDARD*
1153	*SOUND_GARAGE_DOOR_START*
1154	*SOUND_GARAGE_DOOR_STOP*
1163	*SOUND_PED_COLLAPSE*
1165	*SOUND_SHUTTER_DOOR_SLOW_START*
1166	*SOUND_SHUTTER_DOOR_SLOW_STOP*
1169	*SOUND_RESTAURANT_CJ_PUKE*
1183	*SOUND_DRIVING_AWARD_TRACK_START (музыка)*
1184	*SOUND_DRIVING_AWARD_TRACK_STOP*
1185	*SOUND_BIKE_AWARD_TRACK_START (музыка)*
1186	*SOUND_BIKE_AWARD_TRACK_STOP*
1187	*SOUND_PILOT_AWARD_TRACK_START (музыка)*
1188	*SOUND_PILOT_AWARD_TRACK_STOP*
1190	*SOUND_SLAP*
San Andreas Multiplayer 0.3D	
ID	***Название***
3401	*SOUND_OFFICE_FIRE_ALARM (сигнализация для помещений)*
3600	*SOUND_MOBILE_DIALING (набор номера на мобильнике)*
19600	*SOUND_ALARM_CLOCK (звук будильника)*

25800	*SOUND__MECHANIC_ATTACH_CAR_BOMB (установка бомбы в машину)*
17802	*SOUND__GYM_BOXING_BELL (боксерский гонг)*
20600	*SOUND_CAR_PHONE_RING (звонок мобильника)*
20804	*SOUND_PED_MOBRING (звонок мобильника)*
23000	*SOUND_MOBRING (звонок мобильника)*
14800	*SOUND_DETONATION_SIREN (сирена детонации)*
45400	*SOUND_BLIP_DETECTED (сигнал обнаружения)*

ID Интерьеров

Вид Интерьера	Описание	Координаты
Магазины 24/7		
	Магазин 24/7 #1 ID: 17	-25.884499,-185.868988,1003.549988
	Магазин 24/7 #2 ID: 10	-6.091180,-29.271898,1003.549988
	Магазин 24/7 #3 ID: 18	-30.946699,-89.609596,1003.549988
	Магазин 24/7 #4 ID: 16	-25.132599,-139.066986,1003.549988

	Магазин 24/7 #5 ID: 4	-27.312300,-29.277599,1003.549988
	Магазин 24/7 #6 ID: 6	-26.691599,-55.714897,1003.549988
Аэропорты		
	Francis Ticket Sales Airport ID: 14 **Интерьер не доделан.**	-1827.147338,7.207418,1061.143554
	Andromada Cargo Hold ID: 9	315.856170,1024.496459,1949.797363
	Shamal Cabin ID: 1	2.384830,33.103397,1199.849976

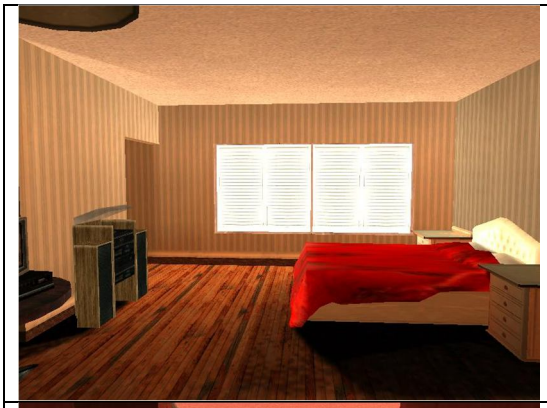
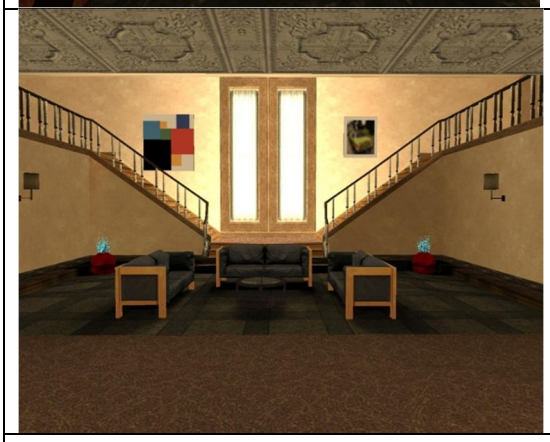
	Abounded AC Tower ID: 10	419.8936, 2537.1155, 10
Аммунация		
	Ammunation #1 ID: 1	286.148987,-40.644398,1001.569946
	Ammunation #2 ID: 4	286.800995,-82.547600,1001.539978
	Ammunation #3 ID: 6	296.919983,-108.071999,1001.569946
	Ammunation #4 ID: 7	314.820984,-141.431992,999.661987

	Ammunation #5 ID: 6	316.524994,-167.706985,999.661987
	Booth Ammunation ID: 7	302.292877,-143.139099,1004.062500
	Range Ammunation ID: 7	280.795104,-135.203353,1004.062500
Дома персонажей одиночной игры		
	B Dup's Apartment ID: 3	1527.0468, -12.0236, 1002.0971
	B Dup's Crack Palace ID: 2 **Интерьер не доделан. Стенки и мебель сквозные.**	1523.5098, -47.8211, 1002.2699

	OG Loc's House ID: 3 **Интерьер не доделан.** **Стенки сквозные.**	512.9291, -11.6929, 1001.565
	Ryder's House ID: 2	2447.8704, -1704.4509, 1013.5078
	Sweet's House ID: 1 **Интерьер не доделан.** **Есть сквозные места.**	2527.0176, -1679.2076, 1015.4986
	Madd Dogg's Mansion ID: 5	1267.8407, -776.9587, 1091.9063
	Big Smoke's Crack Palace ID: 2	2536.5322, -1294.8425, 1044.125

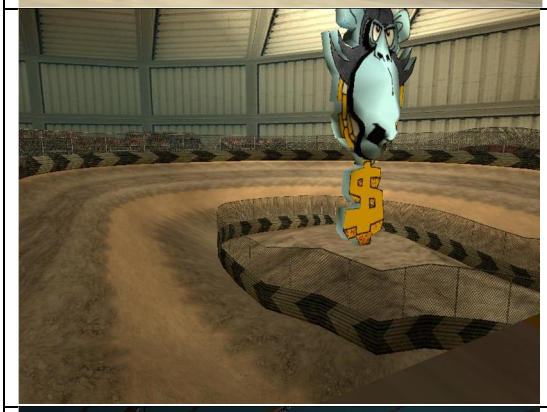
Дома (Safe Houses)		
	CJ's House ID: 3	2496.0549, -1695.1749, 1014.7422
	Angel Pine trailer ID: 2 **Интерьер не доделан.** **Стенки сквозные.**	1.1853, -3.2387, 999.4284, 2
	Safe House #1 ID: 5	2233.6919, -1112.8107, 1050.8828
	Safe House #2 ID: 6	2194.7900, -1204.3500, 1049.0234
	Safe House #3 ID: 9	2319.1272, -1023.9562, 1050.2109

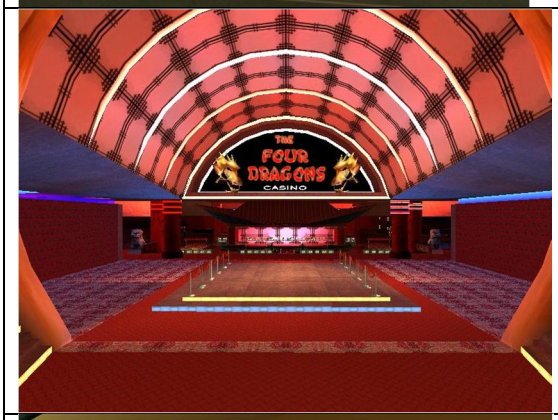
	Safe House #4 ID: 10	2262.4797,-1138.5591,1050.63285
	Safe House #5 ID: 10	2270.39,-1210.45,1046.57
	Safe House #6 ID: 6	2308.79,-1212.88,1048.03
	Safe House #7 ID: 2	2237.59,-1080.87,1048.07
	Verdant Bluff safehouse ID: 8	2365.1089, -1133.0795, 1050.875

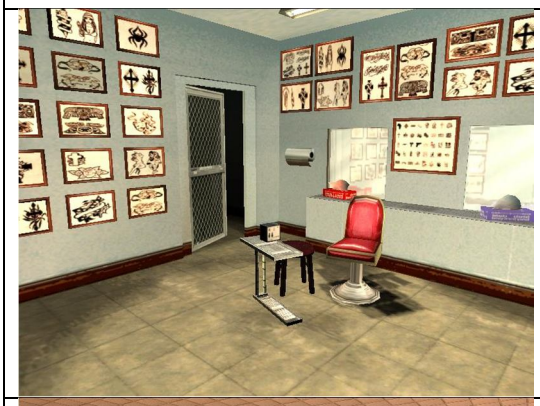
	Willowfield Safehouse ID: 11	2282.9099, -1138.2900, 1050.8984
	he Camel's Toe Safehouse ID: 1	2216.1282, -1076.3052, 1050.4844
	Unused safe house ID: 12	2324.4199,-1145.5683,1050.7100
Интерьеры из миссий		
	Atrium ID: 18	1726.18,-1641.00,20.23

	Burning Desire ID: 5 **Интерьер не доделан.** **Есть сквозные места.**	2338.32,-1180.61,1027.98
	Colonel Furhberger ID: 8	2807.63,-1170.15,1025.57
	Welcome Pump(Dillimore) ID: 1 **Интерьер не доделан.** **Есть сквозные места.**	681.66,-453.32,-25.61
	Woozies Apartment ID: 1	-2158.72,641.29,1052.38
	Jizzy's ID: 3	-2637.69,1404.24,906.46

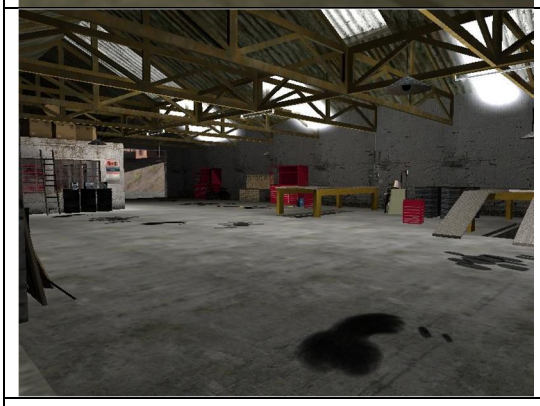
	Dillimore Gas Station ID: 0 **Интерьер не доделан. Есть сквозной пол у выхода**	664.19,-570.73,16.34
	Jefferson Motel ID:15	2220.26,-1148.01,1025.80
	Liberty City ID: 1 **Интерьер не доделан. Есть сквозные места.**	-750.80,491.00,1371.70
	Sherman Dam ID: 17	-944.2402, 1886.1536, 5.0051
Стадионы		

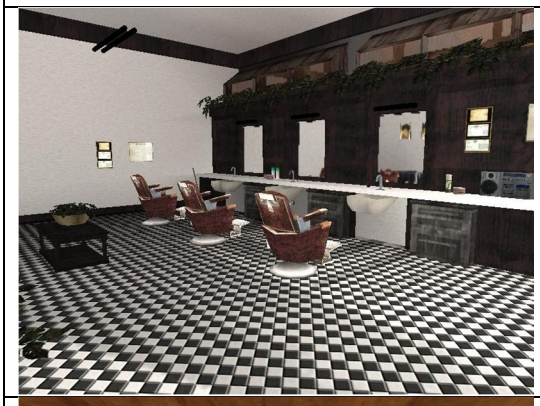
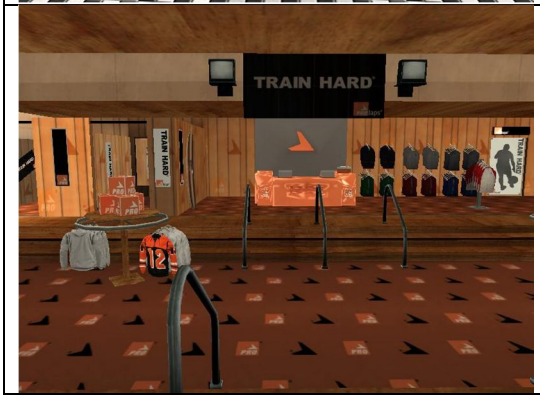
	RC War Arena ID: 10	-1079.99,1061.58,1343.04
	Racing Stadium ID: 7	-1395.958,-208.197,1051.170
	Racing Stadium 2 ID: 4	-1424.9319,-664.5869,1059.8585
	Bloodbowl Stadium ID: 15	-1394.20,987.62,1023.96
	Kickstart Stadium ID: 14	-1410.72,1591.16,1052.53
Казино		

	Caligulas Casino ID: 1	2233.8032,1712.2303,1011.7632
	4 Dragons Casino ID: 10	2016.2699,1017.7790,996.8750
	Redsands Casino ID: 12	1132.9063,-9.7726,1000.6797
	Inside Track betting ID: 3	830.6016, 5.9404, 1004.1797
	Caligulas Roof ID: 1	2268.5156, 1647.7682, 1084.2344

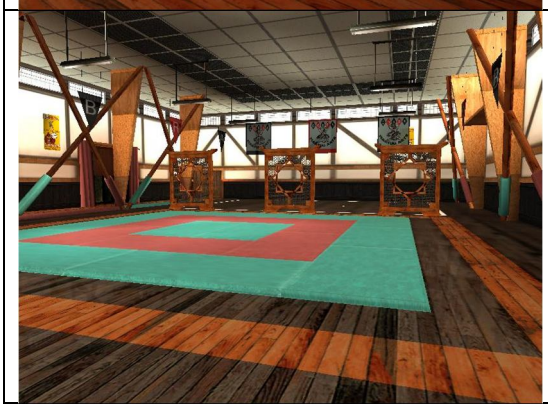
	4 Dragons Janitor's Office ID: 10	1893.0731, 1017.8958, 31.8828
Магазины и забегаловки		
	Tattoo ID: 16	-203.0764,-24.1658,1002.2734
	Burger Shot ID: 10	365.4099,-73.6167,1001.5078
	Well Stacked Pizza ID: 5	372.3520,-131.6510,1001.4922
	Cluckin Bell ID: 9	365.7158,-9.8873,1001.8516

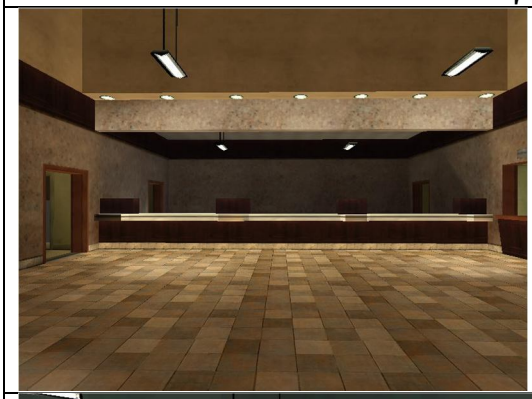
	Rusty Donut's ID: 17	378.026,-190.5155,1000.6328
	Zero's ID: 6	-2240.1028, 136.973, 1035.4141
	Sex Shop ID: 3	-100.2674, -22.9376, 1000.7188
Гаражи		
	Loco Low Co ID: 2	616.7820,-74.8151,997.6350
	Wheel Arch Angels ID: 3	615.2851,-124.2390,997.6350

	Transfender ID: 1	617.5380,-1.9900,1000.6829
	Doherty Garage ID: 1	-2041.2334, 178.3969, 28.8465
Комнаты подруг Карла		
	Denise's Bedroom ID: 1	245.2307, 304.7632, 999.1484
	Helena's Barn ID: 3	290.623, 309.0622, 999.1484
	Barbaras Lovenest ID: 5	322.5014, 303.6906, 999.1484

	Katie's Lovenest ID: 2	269.6405, 305.9512, 999.1484
	Michelle's Lovenest ID: 4	306.1966, 307.819, 1003.3047
	Millie's Bedroom ID: 6	344.9984, 307.1824, 999.1557
Магазины одежды, парикмахерская, шкаф		
	Barber Shop ID: 3	418.4666, -80.4595, 1001.8047
	Pro Laps ID: 3	206.4627, -137.7076, 1003.0938

	Victim ID: 5	225.0306, -9.1838, 1002.218
	Suburban ID: 1	204.1174, -46.8047, 1001.8047
	Reece's Barber Shop ID: 2	414.2987, -18.8044, 1001.8047
	Zip ID: 18	161.4048, -94.2416, 1001.8047
	Didier Sachs ID: 14	204.1658, -165.7678, 1000.5234

	Binco ID: 15	207.5219, -109.7448, 1005.1328
	Barber Shop 2 ID: 12	411.9707, -51.9217, 1001.8984
	Wardrobe ID: 14	256.9047, -41.6537, 1002.0234
Спортзалы		
	Los Santos Gym ID: 5	772.1119,-3.8986,1000.7288
	San Fierro Gym ID: 6	771.8632,-40.5659,1000.6865

	Las Venturas Gym ID: 7	774.0681,-71.8559,1000.6484
Департаменты полиции		
	San Fierro Police Department ID: 10	246.40,110.84,1003.22
	Los Santos Police Department ID: 6	246.6695, 65.8039, 1003.6406
	Las Venturas Police Department ID: 3	288.4723, 170.0647, 1007.1794
	Planning Department (City Hall) ID: 3	386.5259, 173.6381, 1008.382
Дома для грабежа		

	Burglary House #1 ID: 3	234.6087, 1187.8195, 1080.2578
	Burglary House #2 ID: 2	225.5707, 1240.0643, 1082.1406
	Burglary House #3 ID: 1	224.288, 1289.1907, 1082.1406
	Burglary House #4 ID: 5	239.2819, 1114.1991, 1080.9922
	Burglary House #5 ID: 15	295.1391, 1473.3719, 1080.2578

	Burglary House #6 ID: 4	261.1165, 1287.2197, 1080.2578
	Burglary House #7 ID: 10 **Нет мебели!**	24.3769, 1341.1829, 1084.375
	Burglary House #8 ID: 4 **Нет мебели! 2 комнаты**	-262.1759, 1456.6158, 1084.3672
	Burglary House #9 ID: 9 **Нет мебели!**	22.861, 1404.9165, 1084.4297
	Burglary House #10 ID: 5	140.3679, 1367.8837, 1083.8621

	Burglary House #11 ID: 6 **Интерьер в 2 вариантах с мебелью и без мебели**	234.2826, 1065.229, 1084.2101
	Burglary House #12 ID: 6 **Интерьер в 2 вариантах с мебелью и без мебели**	-68.5145, 1353.8485, 1080.2109
	Burglary House #14 ID: 8 **Нет мебели!**	-42.5267, 1408.23, 1084.4297
	Burglary House #15 ID: 9 **Нет мебели!**	84.9244, 1324.2983, 1083.8594
	Burglary House #16 ID: 9 **Нет мебели!**	260.7421, 1238.2261, 1084.2578

	Burglary House #17 ID: 2	447.0000, 1400.3000, 1084.3047
Интерьеры с ID: 0		
	LV Warehouse 1 ID: 0 **Это завод по подделке фишек казино из миссии с Вузи, закрытый.**	1059.8959,2081.6857,10.8203
	SF Bomb Shop ID: 0 **Закрытый гараж по установке бомб на машины из миссий с взрывом завода**	-1685.6364,1035.4761,45.2109
	LS Garage ID: 0 **Закрытый гараж куда нужно было завести фургончик с аудио системой для OG Lock**	1643.8398 -1514.8195 13.5666
	SF Bank ID: 0 **Банк в сельской местности, который нужно было в одной из миссий ограбить**	2315.9528 -1.6181 26.7421

Нет категории		
	Lil' Probe Inn ID: 18	-227.5703, 1401.5544, 27.7656
	Crack Den ID: 5	318.5645, 1118.2079, 1083.8828
	Meat Factory ID: 1	963.0586, 2159.7563, 1011.0303
	Bike School ID: 3	1494.8589, 1306.48, 1093.2953

	Driving School ID: 3	-2031.1196, -115.8287, 1035.1719
Рестораны, клубы		
	Jay's Dinner ID: 4	449.0172, -88.9894, 999.5547
	Big Spread Ranch ID: 3	1212.0762,-28.5799,1000.9531
	The Pig Pen ID: 2	1204.9326,-8.1650,1000.9219

	Dance Club ID: 17	490.2701,-18.4260,1000.6797

Это практически все интерьеры, за исключением тех, которые не пригодны для использования.

ID клавиш для использования в SAMP

Эти ID используются в **OnPlayerKeyStateChange**.

San Andreas Multiplayer 0.3c		
ID	**Назначение клавиши**	**Имя константы**
1	Действие	KEY_ACTION
2	Присесть	KEY_CROUCH
4	Огонь	KEY_FIRE
8	Спринт	KEY_SPRINT
16	Вторичная атака	KEY_SECONDARY_ATTACK
32	Прыжок	KEY_JUMP
64	Смотреть в право	KEY_LOOK_RIGHT
128	Ручной тормоз	KEY_HANDBRAKE
256	Смотреть в небо	KEY_LOOK_LEFT
512	Субмиссия (в машине), Смотреть назад (пешком)	KEY_SUBMISSION KEY_LOOK_BEHIND
1024	Идти шагом	KEY_WALK
2048	Аналог вверх (по умолчанию 8)	KEY_ANALOG_UP
4096	Аналог вниз (по умолчанию 2)	KEY_ANALOG_DOWN
16384	Аналог вправо (по умолчанию 6)	KEY_ANALOG_LEFT
8192	Аналог влево (по умолчанию 4)	KEY_ANALOG_RIGHT
65408	Вперед	KEY_UP
128	Назад	KEY_DOWN
65408	Влево	KEY_LEFT
128	Вправо	KEY_RIGHT
San Andreas Multiplayer 0.3d		
65536	Ответ «Да»	KEY_YES
131072	Ответ «Нет»	KEY_NO

ID Объектов

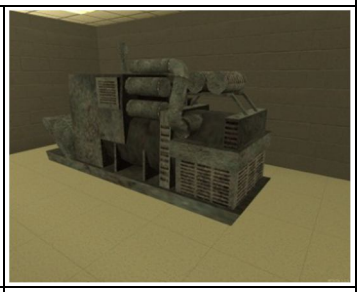
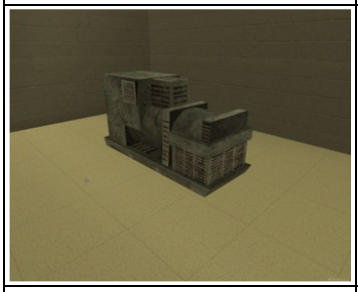
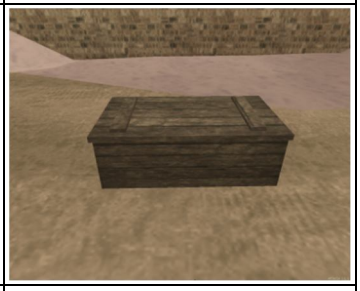
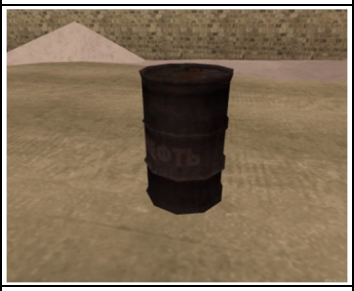
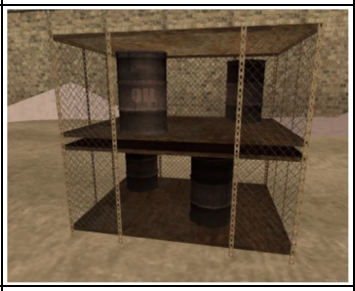
Оружие

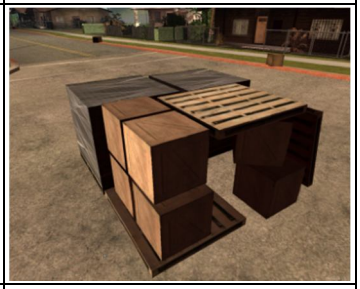
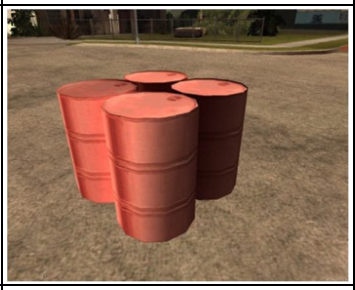
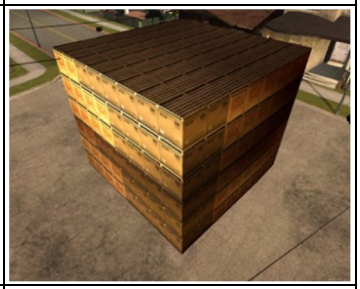
ID объекта	Оружие	ID объекта	Оружие
321	Фиолетовый дилдо	349	Дробовик
322	Маленький белый вибратор	350	Обрез
323	Большой белый вибратор	351	Боевой дробовик
324	Серебряный вибратор	352	Микро СМГ
325	Цветы	353	МП5
326	Трость	354	Световая шашка из истребителя
327	Красная коробочка для кольца	355	АК47
328	Красная коробочка	356	М4
330	Телефон CJ	357	Сельская винтовка
331	Кастет	358	Снайперская винтовка
333	Клюшка для гольфа	359	Ракетная установка
334	Дубинка	360	Самонаводящаяся ракетница
335	Боевой нож	361	Огнемет
336	Баскетбольная бита	362	Миниган
337	Лопата	363	Радиоуправляемый взрывпакет
338	Кий	364	Детонатор
339	Катана	365	Балкончик с краской
341	Бензопила	366	Огнетушитель
342	Граната	367	Камера
343	Слезоточивый газ	368	Очки ночного видения
344	Коктейль Молотова	369	Инфракрасные очки
345	Ракета	370	Джетпак
346	Пистолет кольт 45	371	Парашют
347	Кольт с глушителем	372	Тек9
348	Пустынный орел		

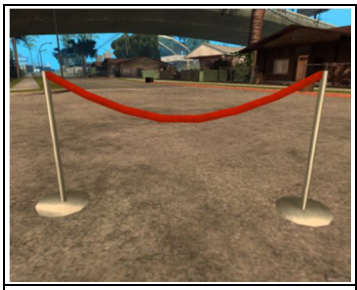
Пикапы

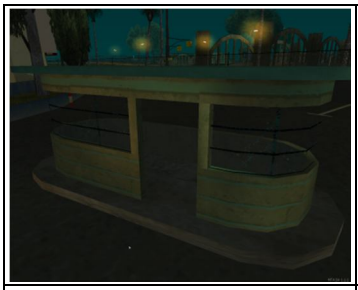
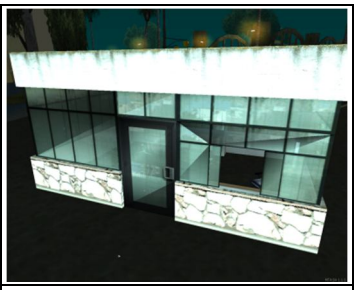
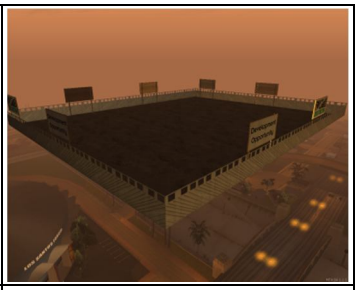
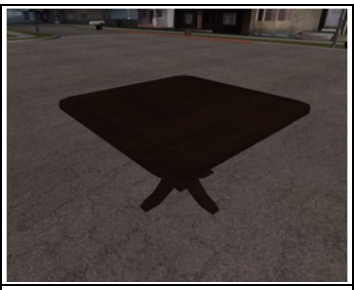
ID объекта	Пикап	ID объекта	Пикап
1240	Красное сердце	1254	Череп
1242	Бронежилет	1274	Знак доллара
1239	Буква I – знак информации	1275	Синяя футболка
1272	Синий домик	1277	Дискета
1273	Зеленый домик	1313	Два черепа
1212	Деньги	1314	Два игрока
1241	Наркотик (красно-белая капсула)	1276	Денежный сверток из Vice City
1247	Значок полиции «Звезда»	1310	Парашют
1248	Логотип GTA3	1318	Белая стрелка – вниз
1252	Бомба из GTA3	1279	Пакет с наркотиками
1253	Знак фотоаппарата	1650	Канистра с бензином
1654	Динамит	1210	Кейс с деньгами

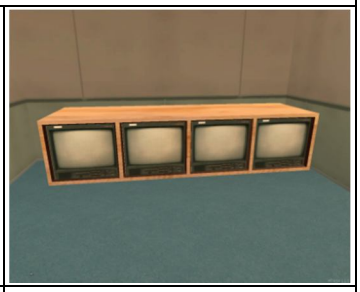
1575	Серый пакет с наркотиками	1576	Оранжевый пакет с наркотиками
1577	Желтый пакет с наркотиками	1578	Зеленый пакет с наркотиками
1579	Синий пакет с наркотиками	1580	Красный пакет с наркотиками
1581	Карточка ученого из Зоны 51	1636	RC бомба

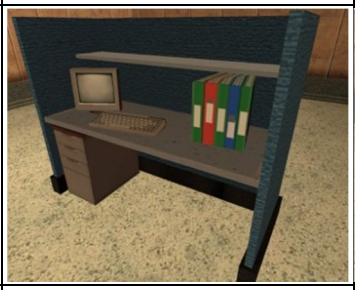
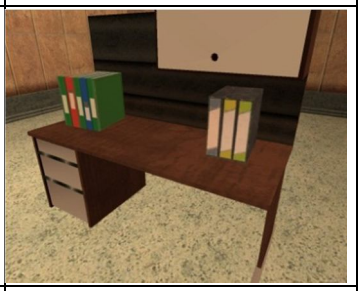
ИНДУСТРИАЛЬНЫЕ ОБЪЕКТЫ			
			
	ГЕНЕРАТОР	ГЕНЕРАТОР	БОЛЬШОЙ ГЕНЕРАТОР
914	**920**	**929**	**934**
			
МАЛЕНЬКИЙ ГЕНЕРАТОР	ЯЩИКИ	ПОЛКА С ЯЩИКАМИ	ПОЛКА С ЯЩИКАМИ
943	**944**	**942**	**939**
			
БОЧКА	ЯЩИКИ	ЯЩИКИ	ЯЩИК
935	**923**	**922**	**2969 / 3052**
			
ЯЩИК	ОРУЖЕЙНЫЙ ЯЩИК	ЯЩИКИ	ЗАБОР НЕ ПАРКОВОЧНАЯ ЗОНА
3013 / 3016	**3014**	**2991**	**2933**
			
БОЧКИ	НЕФТЯНАЯ БОЧКА	СКЛАД НЕФТЯНЫХ БОЧЕК	БАЛОНЫ
2912	**2062**	**925**	**930**

			
ПУСТОЙ СКЛАД БОЧЕК	ЖЕЛЕЗНЫЙ ЯЩИК АРМИИ	ЯЩИК С БОЕПРИПАСАМИ	КОНТЕЙНЕР
931	**964**	**2358**	**2669**
			
ЛЕВАЯ ДВЕРЬ КОНТЕЙНЕРА	ПРАВАЯ ДВЕРЬ КОНТЕЙНЕРА	СКЛАДСКИЕ ЯЩИКИ	СКЛАДСКИЕ ЯЩИКИ
2678	**2679**	**3576**	**3577**
			
СКЛАДСКИЕ ЯЩИКИ	СКЛАДСКАЯ БОЧКА	СКЛАДСКИЕ БОЧКИ	СКЛАДСКИЕ ЯЩИКИ
3630	**3632**	**3633**	**5259**
			
СКЛАДСКИЕ ЯЩИКИ	СКЛАДСКИЕ ЯЩИКИ	ЯЩИК	ВАГОН
5260	**5269**	**1558**	**3585**
			
СКЛАДСКИЕ ЯЩИКИ			
5262			
РАЗНЫЕ ОБЪЕКТЫ			

			
ОГРАДА	ПОСТ ОХРАНЫ АЭРОПОРТА	СТОЛБ АВТОБУСНОЙ СТОЯНКИ	ТАБЛИЧКА
2773	**3881**	**1229**	**1233**
			
ТАБЛИЧКА	НЕБОЛЬШОЕ ПОМЕЩЕНИЕ	ОСНОВАНИЕ ШЛАГБАУМА	ШЛАГБАУМ
1234	**11292**	**966**	**968**
			
АНТЕННА	СВЕРТОК СЕНА	АРХИВ	АРХИВ
3031	**2901**	**2000**	**2007**
			
ДОСКА	ПРИЛАВОК 1	ПРИЛАВОК 2	ПРИЛАВОК 3
3077	**3860**	**3861**	**3862**
			
ПРИЛАВОК 4			
3863	**1340**	**1341**	**1342**

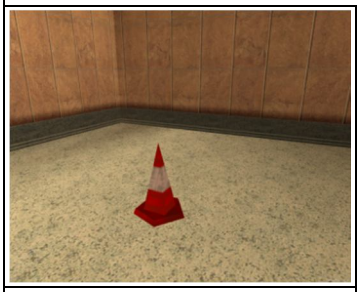
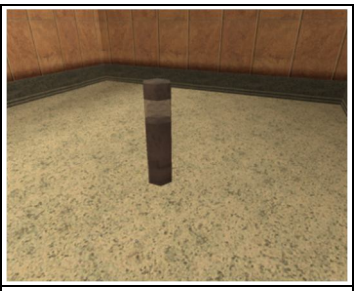
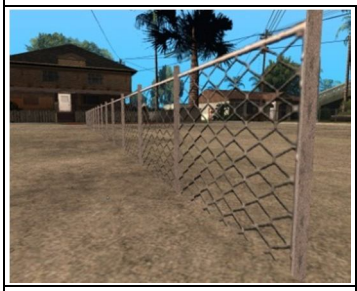
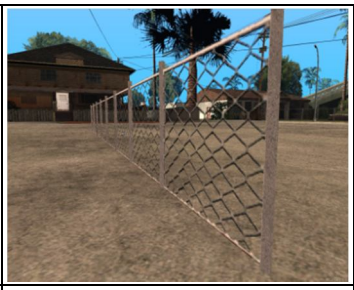
			
ПОСТ ОХРАНЫ	ПОСТ ОХРАНЫ	СТРОЙ ПЛОЩАДКА	ВЫШКА
5837	**8168**	**10394**	**16327**
			
БУДКА	БУДКА СТРОЙ ПЛОЩАДКИ	ПАНЕЛЬ УПРАВЛЕНИЯ	КРЕСЛА ДЛЯ ПУ
967	**1684**	**9819**	**9822**
ВОРОТА			
			
ВОРОТА	ВОРОТА	ВОРОТА	ВОРОТА
969	**971**	**975**	**976**
			
ВОРОТА АЭРОПОРТА	ВОРОТА АЭРОПОРТА	ВОРОТА АЭРОПОРТА	
980	**988**	**989**	
СТОЛЫ			
			
СТОЛ	МАЛЕНЬКИЙ СТОЛ	КРУГЛЫЙ СТОЛ	СТОЛ
2357	**2112**	**2111**	**2236**
ДИВАНЫ			

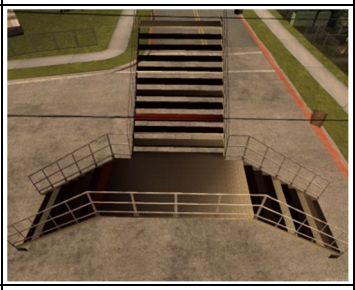
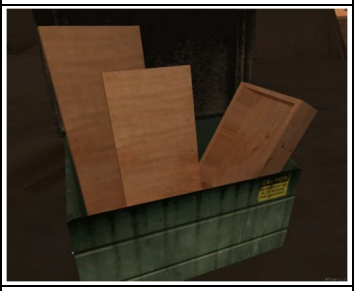
			
ДИВАН 1	ДИВАН 2	ДИВАН 3	ДИВАН 4
1768	**1766**	**1764**	**1763**
			
ДИВАН 5	ДИВАН 6	ДИВАН 7	
1761	**1760**	**1703**	
КРЕСЛА			
			
КРЕСЛО 1	КРЕСЛО 2	КРЕСЛО 3	КРЕСЛО 4
1755	**1759**	**1762**	**1765**
			
КРЕСЛО 5	КРЕСЛО 6		
1767	**1769**		
ОФИС			
			
ПОЛКА	ПОЛКА 2	СТОЛ	ТЕЛЕВИЗОРЫ
2191	**2199**	**2604**	**2606**

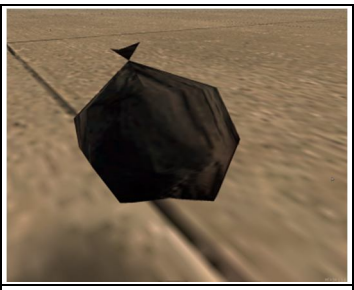
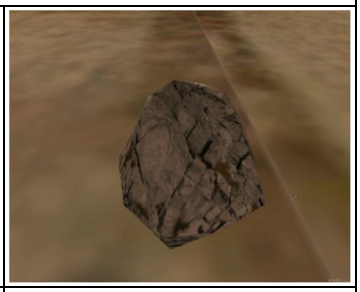
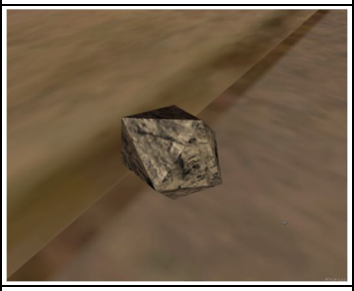
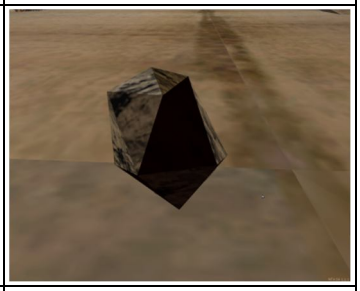
			
ОФИСНЫЙ СТОЛ	ОФИСНЫЙ СТОЛ	ОФИСНЫЙ СТОЛ	ОФИСНЫЙ СТОЛ
1998	**1999**	**2008**	**2009**
			
ОФИСНЫЙ СТОЛ	ОФИСНЫЙ СТОЛ	ОФИСНЫЙ СТОЛ	ОФИСНЫЙ СТОЛ
2165	**2166**	**2172**	**2174**
			
ОФИСНЫЙ СТОЛ	ОФИСНЫЙ СТОЛ	ОФИСНЫЙ СТОЛ	ОФИСНЫЙ СТОЛ
2181	**2182**	**2184**	**2193**
			
ОФИСНЫЙ СТОЛ	ОФИСНЫЙ СТОЛ	ОФИСНЫЙ СТОЛ	ОФИСНЫЙ СТОЛ
2198	**2205**	**2206**	**2308**
			
ОФИСНЫЙ СТОЛ	ОФИСНЫЙ СТОЛ	ОФИСНЫЙ СТУЛ 1	ОФИСНЫЙ СТУЛ 2
2605	**2607**	**1663**	**1671**

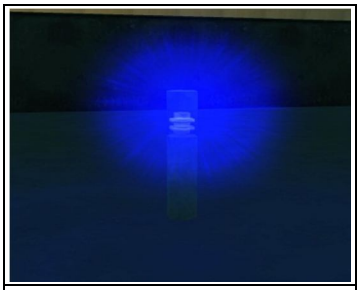
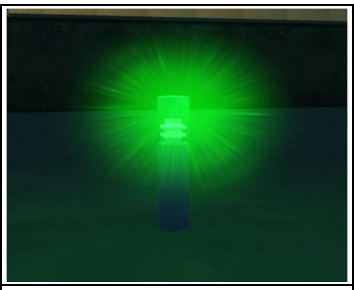
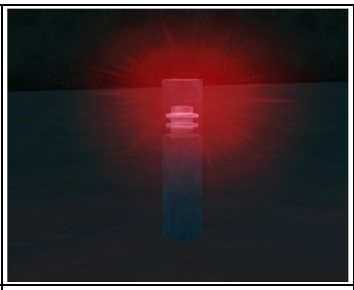
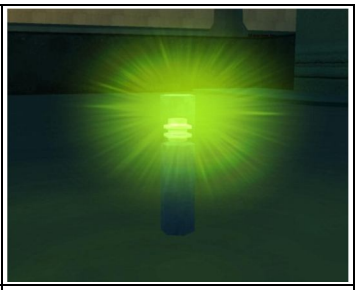
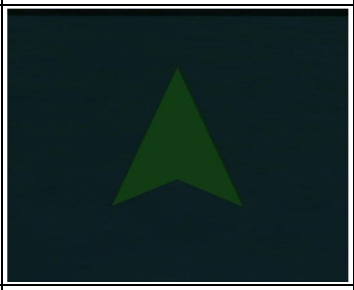
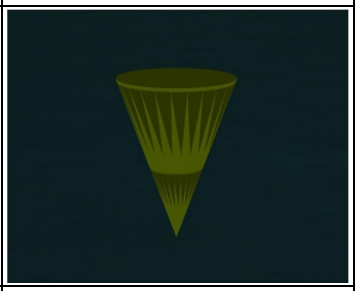
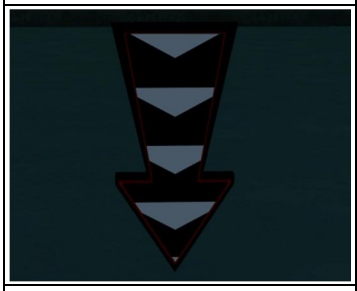
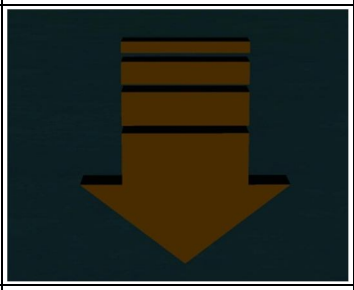
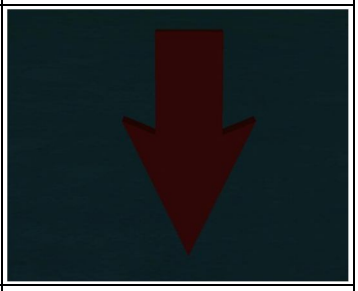
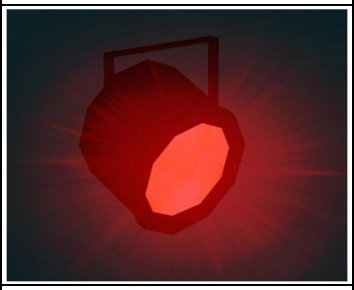
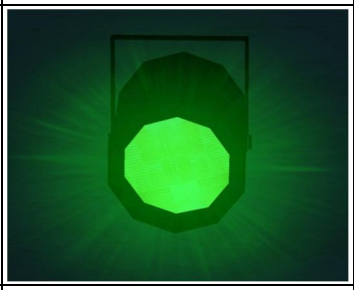
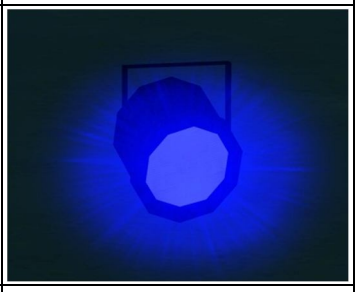
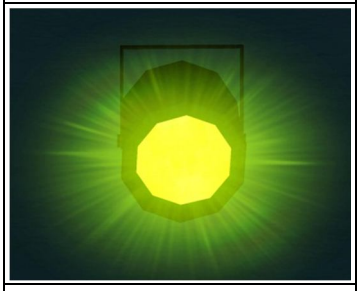
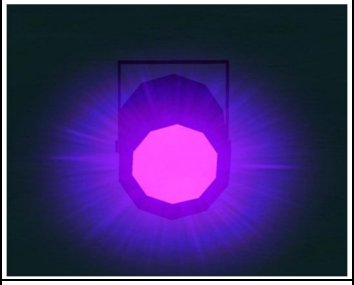
			
ОФИСНЫЙ СТУЛ 3	ОФИСНЫЙ СТУЛ 4	СТУЛ	ОБЫЧНЫЙ СТОЛ
1714	**1715**	**1811**	**2180**
ДВЕРИ			
			
ДВЕРЬ 1	ДВЕРЬ 2	ДВЕРЬ 3	ДВЕРЬ 4
3809	**1491**	**1492**	**1493**
			
ДВЕРЬ 5	ДВЕРЬ 6	ДВЕРЬ 7	ДВЕРЬ 8
1494	**1495**	**1496**	**1497**
			
ДВЕРЬ 9	ДВЕРЬ 10	ДВЕРЬ 11	ДВЕРЬ 12
1498	**1499**	**1500**	**1501**
			
ДВЕРЬ 13	ДВЕРЬ 14	ДВЕРЬ 15	ДВЕРЬ 16
1502	**1504**	**1505**	**1506**

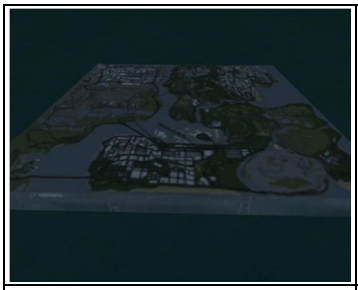
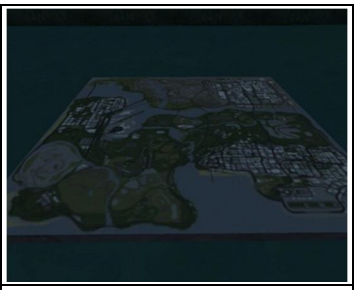
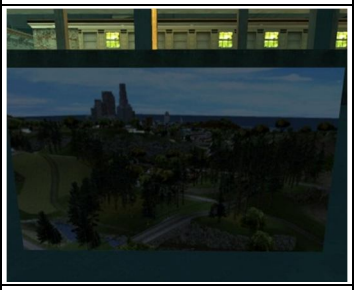
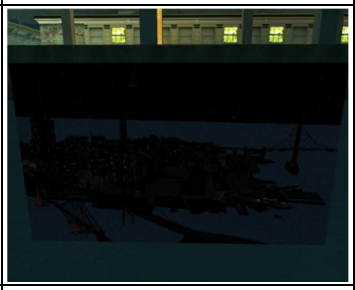
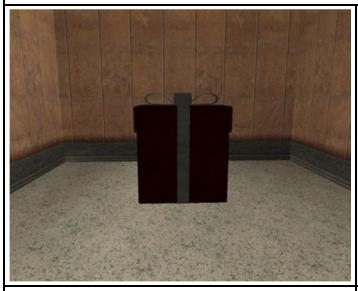
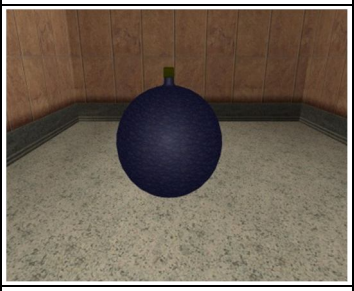
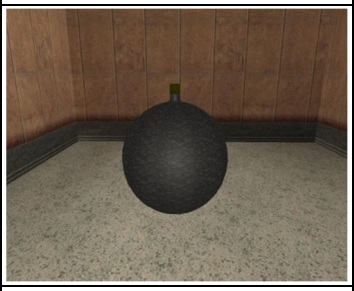
			
ДВЕРЬ 17	ДВЕРЬ 18		
1507	**1508**		
СЛУЖЕБНЫЕ ОБЪЕКТЫ			
			
УЛИЧНЫЙ ТЕЛЕФОН	АВТОБУСНАЯ СТОЯНКА	ПОЧТОВЫЙ ЯЩИК	ТЕЛЕФОННАЯ БУДКА
1216	**1257**	**1258**	**1346**
			
УЛИЧНЫЕ ТЕЛЕФОНЫ	АВТОМАТ	БАНКОМАТ	
1363	**2754**	**2942**	
ГАРАЖНЫЕ ДВЕРИ			
			
ДВЕРЬ 1	ДВЕРЬ 2	ДВЕРЬ 3	ДВЕРЬ 4
7707	**7709**	**7891**	**10184**
			
ДВЕРЬ 5	ДВЕРЬ 6	ДВЕРЬ 7	ДВЕРЬ 8
10558	**2938**	**4084**	**10575**

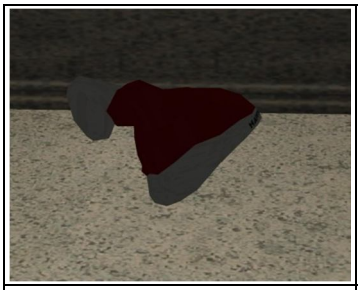
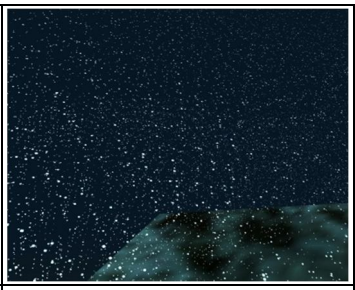
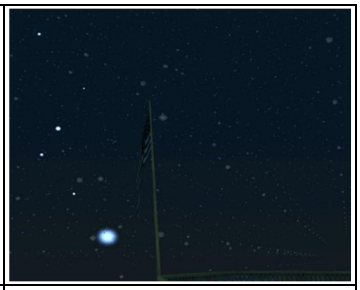
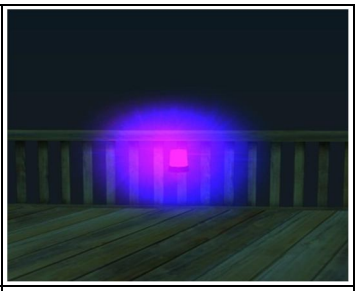
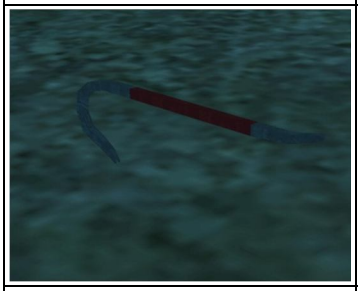
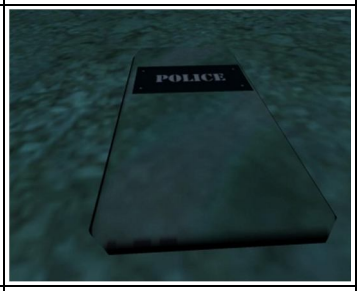
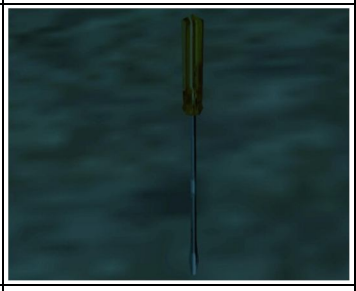
			
ДВЕРЬ 9			
11102			
ДОРОЖНЫЕ ОБЪЕКТЫ			
			
	СТОЛБ		СТОЛБ
1238	**1215**	**1244**	**1319**
			
978	**979**	**1228**	**3091**
ЗАБОРЫ			
			
ЗАБОР 1	ЗАБОР 2	ЗАБОР 3	ЗАБОР 4
982	**983**	**984**	**8167**
			
ЗАБОР 5	ЗАБОР 6	ЗАБОР 7	
8673	**8674**	**4100**	
ЛЕСТНИЦЫ			

			
ЛЕСТНИЦА 1	ЛЕСТНИЦА 2	ЛЕСТНИЦА 3	ЛЕСТНИЦА 4
5816	**5817**	**5820**	**5821**
			
ЛЕСТНИЦА 5	ЛЕСТНИЦА 6	ЛЕСТНИЦА 7	ЛЕСТНИЦА 8
6976	**10244**	**10245**	**7096**
			
ЛЕСТНИЦА 9	ЛЕСТНИЦА 10	ЛЕСТНИЦА 11	ЛЕСТНИЦА 12
1428	**1437**	**5130**	**11544**
МУСОР			
			
МУСОРНЫЙ КОНТЕЙНЕР 1	МУСОРНЫЙ КОНТЕЙНЕР 2	МУСОРНЫЙ КОНТЕЙНЕР 3	МУСОРНЫЙ КОНТЕЙНЕР 4
1331	**1332**	**1333**	**1372**
			
МУСОРНЫЙ КОНТЕЙНЕР 5	МУСОРНЫЙ КОНТЕЙНЕР 6	КОРОБКИ	КОРОБКИ
1227	**1415**	**1440**	**1441**

			
МУСОРНЫЙ ПАКЕТ 1	МУСОРНЫЙ ПАКЕТ 2		
1265	**1264**		
КАМНИ			
			
КАМЕНЬ	ОГРОМНЫЙ КАМЕНЬ 1	ОГРОМНЫЙ КАМЕНЬ 2	МАЛЕНЬКИЙ КАМЕНЬ 1
2936	**897**	**898**	**905**
			
КАМЕНЬ 2	МАЛЕНЬКИЙ КАМЕНЬ 2	МАЛЕНЬКИЙ КАМЕНЬ 3	МАЛЕНЬКИЙ КАМЕНЬ 4
906	**3930**	**3929**	**3931**
РАСТИТЕЛЬНОСТЬ			
			
РАСТЕНИЕ 1	РАСТЕНИЕ 2	РАСТЕНИЕ 3	РАСТЕНИЕ 4
801	**802**	**808**	**809**
			
РАСТЕНИЕ 5	РАСТЕНИЕ 6		
823	**824**		
ИНТЕРЕСНОЕ И НОВОЕ			

			
СИНИЙ ФОНАРЬ	ЗЕЛЕНЫЙ ФОНАРЬ	КРАСНЫЙ ФОНАРЬ	ЖЕЛТЫЙ ФОНАРЬ
19122	**19123**	**19124**	**19125**
			
ГОЛУБОЙ ФОНАРЬ	ФИОЛЕТОВЫЙ ФОНАРЬ	ЗЕЛЕНАЯ СТРЕЛКА	ЖЕЛТЫЙ МАРКЕР
19126	**19127**	**19134**	**19135**
			
ПОЛОСАТАЯ СТРЕЛКА	СИНИЙ УКАЗАТЕЛЬ	ОРАНЖЕВАЯ СТРЕЛКА	КРАСНАЯ СТРЕЛКА
19130	**19131**	**19132**	**19133**
			
МИГАЮЩИЙ БЕЛЫЙ ФОНАРЬ	МИГАЮЩИЙ КРАСНЫЙ ФОНАРЬ	МИГАЮЩИЙ ЗЕЛЕНЫЙ ФОНАРЬ	МИГАЮЩИЙ СИНИЙ ФОНАРЬ
19150	**19151**	**19152**	**19153**
			
МИГАЮЩИЙ ЖЕЛТЫЙ ФОНАРЬ	МИГ. ФИОЛЕТОВЫЙ ФОНАРЬ	МИГАЮЩИЙ ГОЛУБОЙ ФОНАРЬ	ЗЕРКАЛЬНЫЙ ШАР
19154	**19155**	**19156**	**19159**

			
КАРТА ШТАТА	КАРТА ШТАТА	КАРТА ШТАТА	КАРТИНА ШТАТА 2
19164	**19165**	**19166**	**19172**
			
КАРТИНА ШТАТА 2	КАРТИНА ШТАТА 3	КАРТИНА ШТАТА 4	ЛОГОТИП SAMP
19173	**19174**	**19175**	**18749-18750**
НОВОГОДНИЕ ОБЪЕКТЫ (только в SAMP 0.3D)			
			
ПОДАРОК 1	ПОДАРОК 2	ПОДАРОК 3	ПОДАРОК 4
19054	**19055**	**19056**	**19057**
			
ПОДАРОК 5	ЕЛОЧНЫЙ ШАР 1	ЕЛОЧНЫЙ ШАР 2	ЕЛОЧНЫЙ ШАР 3
19058	**19059**	**19060**	**19061**
			
ЕЛОЧНЫЙ ШАР 4	ЕЛОЧНЫЙ ШАР 5	НОВОГОДНЯЯ ШАПКА 1	НОВОГОДНЯЯ ШАПКА 2
19062	**19063**	**19064**	**19065**

			
НОВОГОДНЯЯ ШАПКА 3	ЕЛКА	ПАДАЮЩИЙ СНЕГ	ПАДАЮЩИЙ СНЕГ 2
19066	**19076**	**18863**	**18864**
ДРУГИЕ ОБЪЕКТЫ			
			
ШЛЯПА ШЕРИФА SAPD	ШЛЯПА ШЕРИФА SAPD 2	ПОЛИЦЕЙСКАЯ МИГАЛКА	МОЛОТОК
19099	**19100**	**18646**	**18635**
			
ЛОМ	ГАЕЧНЫЙ КЛЮЧ	ПОЛИЦЕЙСКАЯ КЕПКА	ПОЛИЦЕЙСКИЙ ЩИТ
18634	**18633**	**18636**	**18637**
			
КАСКА	ОТВЕРТКА	МОТОШЛЕМ	ТАКСИ 1
18638	**18644**	**18645**	**19308**
			
ТАКСИ 2	ТАКСИ 3	ТАКСИ 4	ОТКРЫТЫЙ КОНТЕЙНЕР
19309	**19310**	**19311**	**19321**

ID оружия

ID	Иконка	Название оружия	Слот	Модель
0		Кулак	0	-
1		Кастет	0	331
2		Клюшка для гольфа	1	333
3		Полицейская дубинка	1	334
4		Нож	1	335
5		Бейсбольная бита	1	336
6		Лопата	1	337
7		Кий	1	338
8		Катана	1	339
9		Бензопила	1	341
10		Фиолетовый дилдо	10	321
11		Маленький белый вибратор	10	322
12		Большой белый вибратор	10	323
13		Серебряный вибратор	10	324
14		Цветы	10	325
15		Трость	10	326
16		Граната	8	342
17		*Слезоточивый газ	8	343
18		Коктейль Молотова	8	344
22		Кольт	2	346
23		Кольт с глушителем	2	347
24		Пустынный орел	2	348
25		Дробовик	3	349
26		Обрез	3	350
27		Боевой дробовик	3	351
28		Микро СМГ	4	352
29		МП5	4	353
30		АК-47	5	355
31		М4	5	356
32		Тек9	4	372
33		Сельская винтовка	6	357

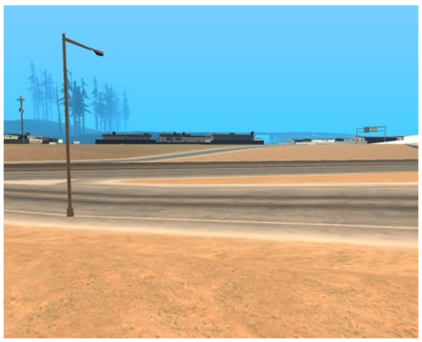
34		Снайперская винтовка	6	358
35		Ракетная установка	7	359
36		Самонаводящаяся ракетница	7	360
37		Огнемет	7	361
38		Миниган	7	362
39		Радиоуправляемый взрывпакет	8	363
40		Детонатор	12	364
41		Баллончик с краской	9	365
42		Огнетушитель	9	366
43		Камера	9	367
44		Очки ночного видения	11	368
45		Очки термального видения	11	369
46		Парашют		371
		ID причины смерти игрока		
47		Fake Pistol	-	-
49		Задавила машина	-	-
50		Попал под лопасти вертолета	-	-
51		Взрыв	-	-
53		Утонул	-	-
54		Упал с высоты	-	-
		ID статуса		
200		Присоединился к серверу	-	-
201		Отключился от сервера	-	-

** - Эффект от слезоточивого газа отключен в SA-MP.*

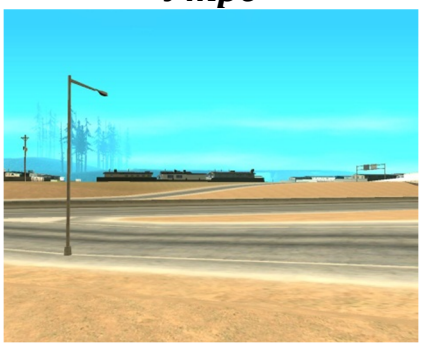
ID погоды

Эти ID нужно использовать в функции **SetWeather**

Утро

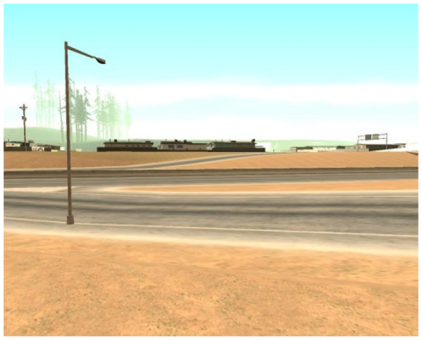


Weather ID: *0*

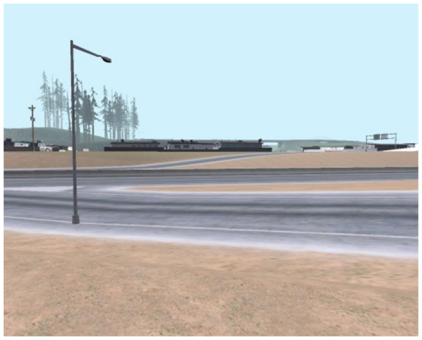


Vehicle ID: *1*

Vehicle ID: *2*

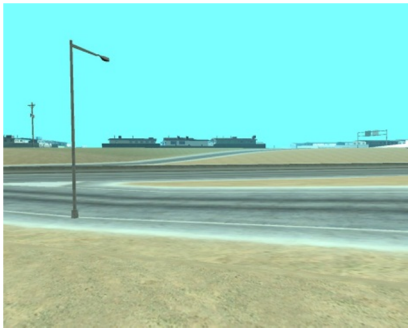


Weather ID: *3*



Vehicle ID: *4*

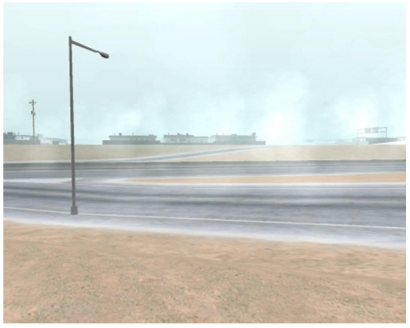
Vehicle ID: *5*



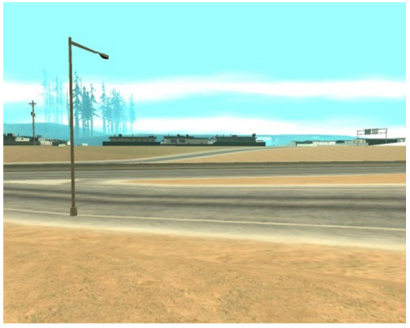
Weather ID: *6*

Vehicle ID: *7*

Vehicle ID: *8*



Weather ID: *9*



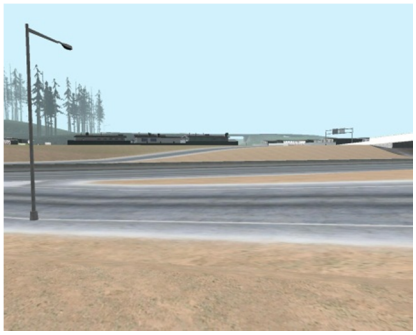
Vehicle ID: *10*

Vehicle ID: *11*

Weather ID: *12*

Vehicle ID: *13*

Vehicle ID: *14*



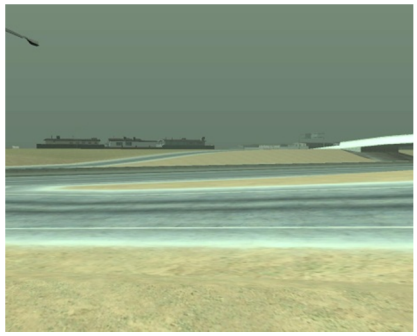
Weather ID: *15*

Weather ID: *16*

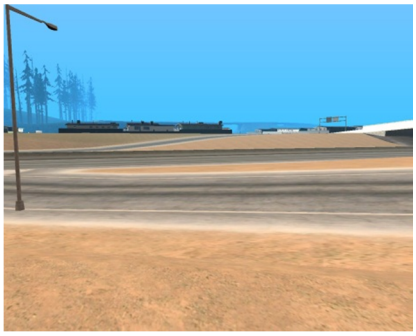
Weather ID: *17*

Weather ID: *18*

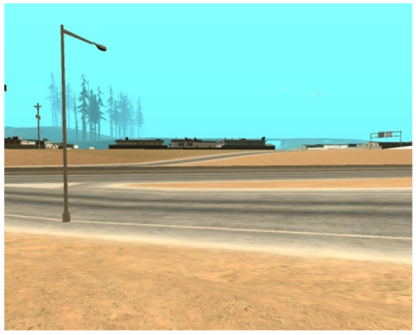
Weather ID: *19*



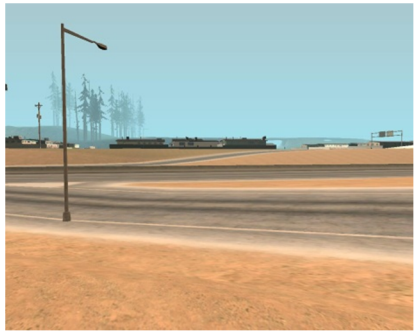
Weather ID: *20*



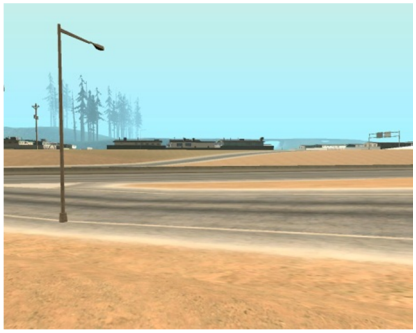
Weather ID: *23*



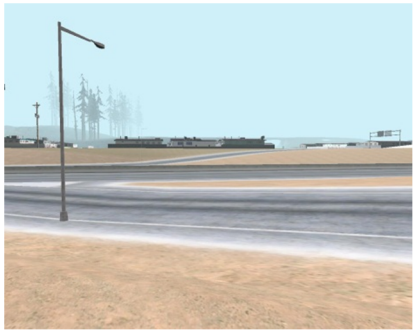
Weather ID: *24*



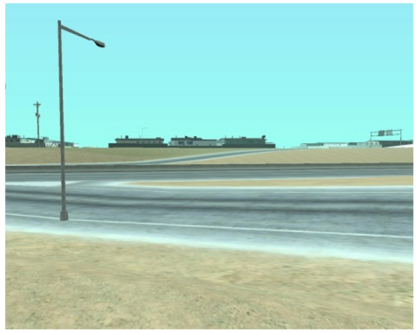
Weather ID: *25*



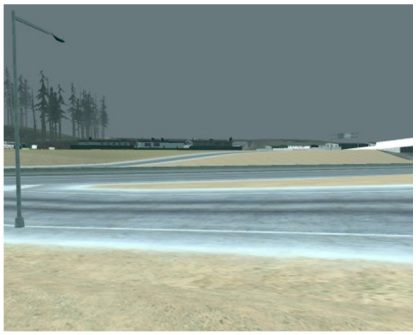
Weather ID: *26*



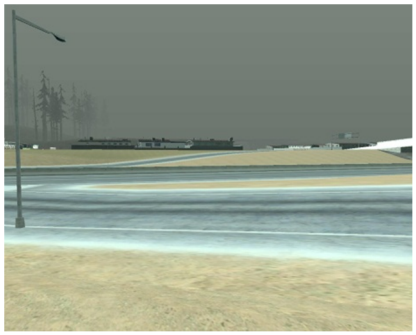
Weather ID: *27*



Weather ID: *28-29*

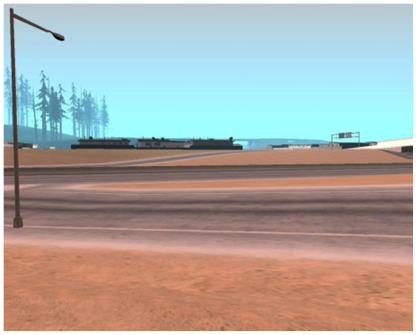


Weather ID: *30*



Weather ID: *31*

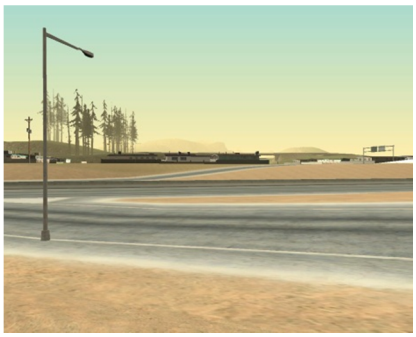
Weather ID: *32*



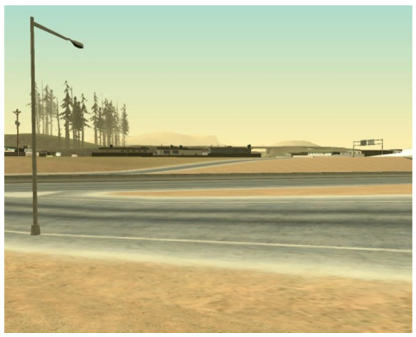
Weather ID: *33*

Weather ID: *34*

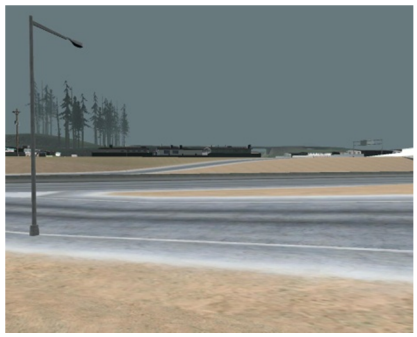
Weather ID: *35*



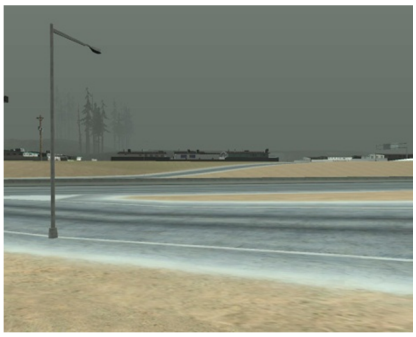
Weather ID: *36*



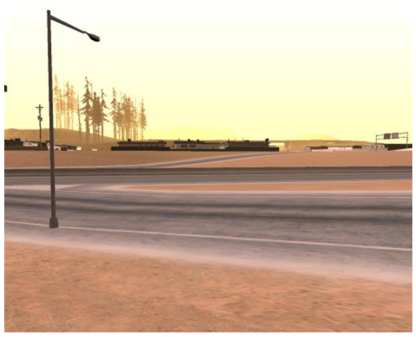
Weather ID: *37*



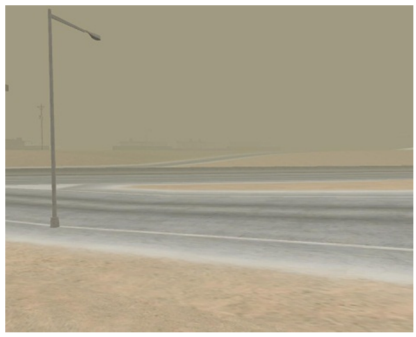
Weather ID: *38*



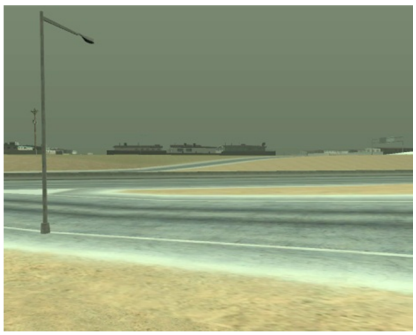
Weather ID: *39*



Weather ID: *40-41*



Weather ID: *42*



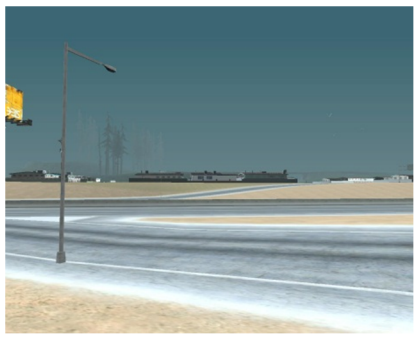
Weather ID: *43*

Weather ID: *46*

Weather ID: *47*

Weather ID: *48*

Weather ID: *49*



Weather ID: *50*

Это не все ID погоды, пропущенные ID погоды, которые в реальности не существуют. Есть погода с ID 2009, это все равно что штат без неба на розоватом фоне, вода становится прозрачной, а если подняться в воздух, как можно выше, весь штат Сан Андреас будет виден как на ладони, как показано на скриншотах ниже:

Weather ID: *2009*

Кроме этого, транспорт и сам персонаж будут уже видны в других цветовых оттенках.
ID дождливой погоды (8 и 16).

ID покрасочных работ машин

Используйте их для автовызываемой функции: **OnVehiclePaintjob**.

ID покрасочной работы: 0

Для автомобиля: Camper

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 1

Для автомобиля: Remington

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 2

Для автомобиля: Remington

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 0

Для автомобиля: Slamvan

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 1

Для автомобиля: Slamvan

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 2

Для автомобиля: Blade

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 0

Для автомобиля: Blade

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 1

Для автомобиля: Slamvan

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 2

Для автомобиля: Blade

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 0

Для автомобиля: Uranus

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 1

Для автомобиля: Uranus

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 2

Для автомобиля: Uranus

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 0

Для автомобиля: Jester

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 1

Для автомобиля: Jester

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 2

Для автомобиля: Jester

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 0

Для автомобиля: Sultan

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 1

Для автомобиля: Sultan

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 2

Для автомобиля: Sultan

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 0

Для автомобиля: Stratum

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 1

Для автомобиля: Stratum

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 2

Для автомобиля: Stratum

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 0

Для автомобиля: Elegy

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 1

Для автомобиля: Elegy

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 2

Для автомобиля: Elegy

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 0

Для автомобиля: Flash

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 1

Для автомобиля: Flash

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 2

Для автомобиля: Flash

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 0

Для автомобиля: Savanna

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 1

Для автомобиля: Savanna

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 2

Для автомобиля: Savanna

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 0

Для автомобиля: Broadway

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 1

Для автомобиля: Broadway

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 0

Для автомобиля: Tornado

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 1

Для автомобиля: Tornado

Цвет: Стандартный (без перекраски в тюнинге)

ID покрасочной работы: 2

Для автомобиля: Tornado

Цвет: Стандартный (без перекраски в тюнинге)

ID Режимов камеры игрока

ID	***Имя встроенной константы заменяющей ID режима камеры***	***Описание вида камеры***
0	*CAMERA_MODE_NONE*	
3	*CAMERA_MODE_INTRAIN*	*Режим камеры при посадке на поезд.*
4	*CAMERA_MODE_ONFOOT*	*Нормальная камера (5,6 другие позиции)*
7	*CAMERA_MODE_AIM_SNIPER*	*Прицел снайпера.*
8	*CAMERA_MODE_AIM_ROCKETLAUNCHER*	*Прицел ракетной установки.*
14	*CAMERA_MODE_FIXED_TARGET*	
15	*CAMERA_MODE_FIXED_POS*	
16	*CAMERA_MODE_FIXED_CARBUMPER*	*Камера на бампере машины*
18	*CAMERA_MODE_INCAR*	*Нормальная камера в машине (19,20,21 другие режимы этой камеры)*
22	*CAMERA_MODE_INBOAT*	*Режим камеры когда игрок на катере.*
46	*CAMERA_MODE_AIM_CAMERA*	*Режим камеры при съемке фотокамерой.*
51	*CAMERA_MODE_AIM_HEATSEEKER*	
53	*CAMERA_MODE_AIM*	*Режим камеры при прицеливании из любого другого вида оружия.*
55	*CAMERA_MODE_DRIVEBY*	*Режим камеры при прицеливании пассажира транспорта.*
56	*CAMERA_MODE_FIXED_CINEMA*	
57	*CAMERA_MODE_FIXED_CINEMA_FLYOVER*	
58	*CAMERA_MODE_FIXED_CINEMA_ZOOM*	
62	*CAMERA_MODE_FIXED_CINEMA_SIDE*	
63	*CAMERA_MODE_FIXED_CINEMA_ROUND*	
64	*CAMERA_MODE_FIXED_CINEMA_HEADON*	

ID Скинов

San Andreas Multiplayer 0.3C


Skin ID: 0


Skin ID: 1


Skin ID: 2


Skin ID: 7


Skin ID: 9


Skin ID: 10


Skin ID: 11


Skin ID: 12


Skin ID: 13


Skin ID: 14


Skin ID: 15


Skin ID: 16


Skin ID: 17


Skin ID: 18


Skin ID: 19


Skin ID: 20


Skin ID: 21


Skin ID: 22


Skin ID: 23


Skin ID: 24


Skin ID: 25


Skin ID: 26


Skin ID: 27


Skin ID: 28


Skin ID: 29


Skin ID: 30


Skin ID: 31


Skin ID: 32


Skin ID: 33


Skin ID: 34


Skin ID: 35


Skin ID: 36


Skin ID: 37


Skin ID: 38


Skin ID: 39


Skin ID: 40


Skin ID: 41


Skin ID: 43


Skin ID: 44


Skin ID: 45


Skin ID: 46


Skin ID: 47


Skin ID: 48


Skin ID: 49


Skin ID: 50


Skin ID: 51


Skin ID: 52


Skin ID: 54


Skin ID: 55


Skin ID: 56


Skin ID: 57


Skin ID: 58


Skin ID: 59


Skin ID: 60


Skin ID: 61


Skin ID: 62


Skin ID: 63


Skin ID: 64


Skin ID: 66


Skin ID: 67


Skin ID: 68

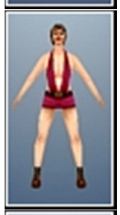

Skin ID: 69


Skin ID: 70


Skin ID: 71


Skin ID: 72


Skin ID: 73


Skin ID: 75


Skin ID: 76


Skin ID: 77


Skin ID: 78


Skin ID: 79


Skin ID: 80


Skin ID: 81


Skin ID: 82


Skin ID: 83


Skin ID: 84

Skin ID: 85

Skin ID: 87

Skin ID: 88

Skin ID: 89

Skin ID: 90

Skin ID: 92

Skin ID: 93

Skin ID: 94

Skin ID: 95

Skin ID: 96

Skin ID: 97

Skin ID: 98

Skin ID: 99

Skin ID: 100

Skin ID: 101

Skin ID: 102

Skin ID: 103

Skin ID: 104

Skin ID: 105

Skin ID: 106

Skin ID: 107

Skin ID: 108

Skin ID: 109

Skin ID: 110

Skin ID: 111

Skin ID: 112

Skin ID: 113

Skin ID: 114

Skin ID: 115

Skin ID: 116

Skin ID: 117

Skin ID: 118

Skin ID: 120

Skin ID: 121

Skin ID: 122

Skin ID: 123

Skin ID: 124

Skin ID: 125

Skin ID: 126

Skin ID: 127


Skin ID: 128


Skin ID: 129


Skin ID: 130


Skin ID: 131


Skin ID: 132


Skin ID: 133


Skin ID: 134


Skin ID: 135


Skin ID: 136


Skin ID: 137


Skin ID: 138


Skin ID: 139


Skin ID: 140


Skin ID: 141


Skin ID: 142


Skin ID: 143


Skin ID: 144


Skin ID: 145


Skin ID: 146


Skin ID: 147

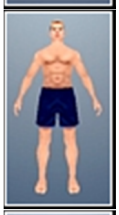

Skin ID: 148


Skin ID: 150


Skin ID: 151


Skin ID: 152


Skin ID: 153


Skin ID: 154


Skin ID: 155


Skin ID: 156


Skin ID: 157


Skin ID: 158


Skin ID: 159


Skin ID: 160


Skin ID: 161


Skin ID: 162


Skin ID: 163


Skin ID: 164


Skin ID: 165


Skin ID: 166


Skin ID: 167


Skin ID: 168


Skin ID: 169


Skin ID: 170


Skin ID: 171


Skin ID: 172

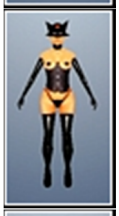

Skin ID: 173


Skin ID: 174


Skin ID: 175


Skin ID: 176


Skin ID: 177


Skin ID: 178


Skin ID: 179


Skin ID: 180


Skin ID: 181


Skin ID: 182


Skin ID: 183


Skin ID: 184


Skin ID: 185


Skin ID: 186


Skin ID: 187


Skin ID: 188


Skin ID: 189


Skin ID: 190


Skin ID: 191


Skin ID: 192


Skin ID: 193


Skin ID: 194


Skin ID: 195


Skin ID: 196


Skin ID: 197


Skin ID: 198


Skin ID: 199


Skin ID: 200


Skin ID: 201


Skin ID: 202


Skin ID: 203


Skin ID: 204


Skin ID: 205


Skin ID: 206


Skin ID: 207


Skin ID: 209


Skin ID: 210


Skin ID: 211


Skin ID: 212


Skin ID: 213


Skin ID: 214


Skin ID: 215


Skin ID: 216


Skin ID: 217


Skin ID: 218


Skin ID: 219


Skin ID: 220


Skin ID: 221


Skin ID: 222


Skin ID: 223


Skin ID: 224


Skin ID: 225


Skin ID: 226


Skin ID: 227


Skin ID: 228


Skin ID: 229


Skin ID: 230


Skin ID: 231


Skin ID: 232


Skin ID: 233


Skin ID: 234


Skin ID: 235


Skin ID: 236


Skin ID: 237


Skin ID: 238


Skin ID: 239


Skin ID: 240


Skin ID: 241


Skin ID: 242


Skin ID: 243


Skin ID: 244


Skin ID: 245


Skin ID: 246

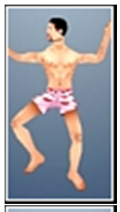

Skin ID: 247


Skin ID: 248


Skin ID: 249


Skin ID: 250


Skin ID: 251


Skin ID: 252


Skin ID: 253


Skin ID: 254


Skin ID: 255


Skin ID: 256


Skin ID: 257


Skin ID: 258


Skin ID: 259


Skin ID: 260


Skin ID: 261


Skin ID: 262


Skin ID: 263


Skin ID: 264


Skin ID: 265


Skin ID: 266


Skin ID: 267


Skin ID: 268


Skin ID: 269


Skin ID: 270


Skin ID: 271


Skin ID: 272


Skin ID: 274


Skin ID: 275


Skin ID: 276


Skin ID: 277


Skin ID: 278


Skin ID: 279


Skin ID: 280


Skin ID: 281


Skin ID: 282


Skin ID: 283


Skin ID: 284


Skin ID: 285


Skin ID: 286


Skin ID: 287


Skin ID: 288


Skin ID: 290


Skin ID: 291


Skin ID: 292


Skin ID: 293


Skin ID: 294


Skin ID: 295


Skin ID: 296


Skin ID: 297


Skin ID: 298


Skin ID: 299

San Andreas Multiplayer 0.3D


Skin ID: 3


Skin ID: 4


Skin ID: 5


Skin ID: 6


Skin ID: 8


Skin ID: 42


Skin ID: 65


Skin ID: 86


Skin ID: 119


Skin ID: 149


Skin ID: 208


Skin ID: 273


Skin ID: 289

ID специальных действий

Эти ID используются

0 - SPECIAL_ACTION_NONE	Пустое действие (остановить анимацию)
1 - SPECIAL_ACTION_DUCK	Наклон
2 - SPECIAL_ACTION_USEJETPACK	Использовать джетпак
3 - SPECIAL_ACTION_ENTER_VEHICLE	Анимация посадки в транспорт
4 - SPECIAL_ACTION_EXIT_VEHICLE	Анимация выхода из транспорта
5 - SPECIAL_ACTION_DANCE1	Танец 1
6 - SPECIAL_ACTION_DANCE2	Танец 2
7 - SPECIAL_ACTION_DANCE3	Танец 3
8 - SPECIAL_ACTION_DANCE4	Танец 4 (Стриптиз)
10 - SPECIAL_ACTION_HANDSUP	Руки вверх
11 - SPECIAL_ACTION_USECELLPHONE	Использование телефона
12 - SPECIAL_ACTION_SITTING	Присесть
13 - SPECIAL_ACTION_STOPUSECELLPHONE	Кладёт телефон обратно
20 - SPECIAL_ACTION_DRINK_BEER	Пьёт пиво (повышается уровень опьянения)
21 - SPECIAL_ACTION_SMOKE_CIGGY	Курит сигару
22 - SPECIAL_ACTION_DRINK_WINE	Пьёт вино
23 - SPECIAL_ACTION_DRINK_SPRUNK	Пьёт Sprunk из алюминиевой банки
68 - SPECIAL_ACTION_PISSING	Справляет малую нужду (нет в a_samp.inc)

ID Типов взрывов

ID	Видимость	Взрывная волна	Создает огонь	Физический урон	Особенность	Радиус поражения
0	Да	Нет	Нет	Да	Нормальный взрыв	Большой
1	Да	Нет	Да	Нет	Нормальный взрыв	Средний
2	Да	Нет	Да	Да	Нормальный взрыв	Большой
3	Да	Нет	Иногда	Да	Нормальный взрыв	Большой
4	Да	Да	Нет	Да	Необычный взрыв, наносит физический урон, без звука. FX эффект.	Средний
5	Да	Да	Нет	Да	Необычный взрыв, наносит физический урон, без звука. FX эффект.	Средний
6	Да	Нет	Нет	Да	Дополнительное красноватое свечение после взрыва	Очень большой
7	Да	Нет	Нет	Да	Дополнительное красноватое свечение после взрыва	Огромный
8	Нет	Нет	Нет	Да	Невидимый	Средний
9	Нет	Нет	Да	Да	Взрыв слышно, но не видно. Создает огонь на уровне земли!	Средний
10	Да	Нет	Нет	Да	Нормальный взрыв	Большой
11	Да	Нет	Нет	Да	Нормальный взрыв	Маленький
12	Да	Нет	Нет	Да	Маленький размер взрыва	Очень маленький
13	Нет	Нет	Нет	Нет	Не производит ни каких специальных эффектов. Оставляет черное пятно на поверхности, не наносит урона!	Большой

ID транспорта

Машины

Vehicle ID: 400
Vehicle Name: Landstalker
Modifications: Transfender

Vehicle ID: 401
Vehicle Name: Bravura
Modifications: Transfender

Vehicle ID: 402
Vehicle Name: Buffalo
Modifications: Transfender

Vehicle ID: 403
Vehicle Name: Linerunner
Modifications: None

Vehicle ID: 404
Vehicle Name: Perenniel
Modifications: Transfender

Vehicle ID: 405
Vehicle Name: Sentinel
Modifications: Transfender

Vehicle ID: 406
Vehicle Name: Dumper
Modifications: None

Vehicle ID: 407
Vehicle Name: Firetruck
Modifications: None

Vehicle ID: 408
Vehicle Name: Trashmaster
Modifications: None

Vehicle ID: 409
Vehicle Name: Stretch
Modifications: Transfender

Vehicle ID: 410
Vehicle Name: Manana
Modifications: Transfender

Vehicle ID: 411
Vehicle Name: Infernus
Modifications: Transfender

Vehicle ID: 412
Vehicle Name: Voodoo
Modifications: Loco Low Co

Vehicle ID: 413
Vehicle Name: Pony
Modifications: None

Vehicle ID: 414
Vehicle Name: Mule
Modifications: None

Vehicle ID: 415
Vehicle Name: Cheetan
Modifications: Transfender

Vehicle ID: 416
Vehicle Name: Ambulance
Modifications: None

Vehicle ID: 418
Vehicle Name: Moonbeam
Modifications: Transfender

Vehicle ID: 419
Vehicle Name: Esperanto
Modifications: Transfender

Vehicle ID: 420
Vehicle Name: Taxi
Modifications: Transfender

Vehicle ID: 421
Vehicle Name: Washington
Modifications: Transfender

Vehicle ID: 422
Vehicle Name: Bobcat
Modifications: Transfender

Vehicle ID: 423
Vehicle Name: Mr Whoopee
Modifications: None

Vehicle ID: 424
Vehicle Name: BF Injection
Modifications: Transfender

Vehicle ID: 426
Vehicle Name: Premier
Modifications: Transfender

Vehicle ID: 427
Vehicle Name: Enforcer
Modifications: None

Vehicle ID: 428
Vehicle Name: Securicar
Modifications: None

Vehicle ID: 429
Vehicle Name: Banshee
Modifications: Transfender

Vehicle ID: 431
Vehicle Name: Bus
Modifications: None

Vehicle ID: 432
Vehicle Name: Rhino
Modifications: None

Vehicle ID: 433
Vehicle Name: Barracks
Modifications: None

Vehicle ID: 434
Vehicle Name: Hotknife
Modifications: None

Vehicle ID: 436
Vehicle Name: Previon
Modifications: Transfender

Vehicle ID: 437
Vehicle Name: Coach
Modifications: None

Vehicle ID: 438
Vehicle Name: Cabbie
Modifications: Transfender

Vehicle ID: 439
Vehicle Name: Stallion
Modifications: Transfender

Vehicle ID: 440
Vehicle Name: Rumpo
Modifications: None

Vehicle ID: 442
Vehicle Name: Romero
Modifications: Transfender

Vehicle ID: 443
Vehicle Name: Packer
Modifications: None

Vehicle ID: 444
Vehicle Name: Monster
Modifications: None

Vehicle ID: 445
Vehicle Name: Admiral
Modifications: Transfender

Vehicle ID: 449
Vehicle Name: Tram
Modifications: None

Vehicle ID: 451
Vehicle Name: Turismo
Modifications: Transfender

Vehicle ID: 455
Vehicle Name: Flatbed
Modifications: None

Vehicle ID: 456
Vehicle Name: Yankee
Modifications: None

Vehicle ID: 457
Vehicle Name: Caddy
Modifications: None

Vehicle ID: 458
Vehicle Name: Solair
Modifications: Transfender

Vehicle ID: 459
Vehicle Name: Berkley's RC Van
Modifications: None

Vehicle ID: 466
Vehicle Name: Glendale
Modifications: Transfender

Vehicle ID: 467
Vehicle Name: Oceanic
Modifications: Transfender

Vehicle ID: 470
Vehicle Name: Patriot
Modifications: None

Vehicle ID: 474
Vehicle Name: Hermes
Modifications: Transfender

Vehicle ID: 475
Vehicle Name: Sabre
Modifications: Transfender

Vehicle ID: 477
Vehicle Name: ZR-350
Modifications: Transfender

Vehicle ID: 478
Vehicle Name: Walton
Modifications: Transfender

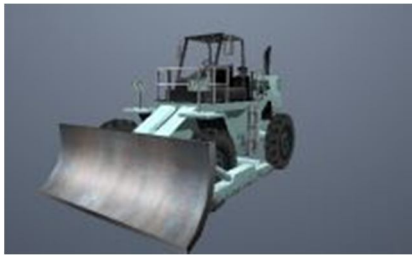
Vehicle ID: 479
Vehicle Name: Regina
Modifications: Transfender

Vehicle ID: 480
Vehicle Name: Comet
Modifications: Transfender

Vehicle ID: 482
Vehicle Name: Burrito
Modifications: None

Vehicle ID: 483
Vehicle Name: Camper
Modifications: None

Vehicle ID: 485
Vehicle Name: Baggage
Modifications: None



Vehicle ID: 486
Vehicle Name: Dozer
Modifications: None

Vehicle ID: 489
Vehicle Name: Dozer
Modifications: Transfender

Vehicle ID: 490
Vehicle Name: FBI Rancher
Modifications: None

Vehicle ID: 491
Vehicle Name: Virgo
Modifications: Transfender

Vehicle ID: 492
Vehicle Name: Greenwood
Modifications: Transfender

Vehicle ID: 494
Vehicle Name: Hotring Racer
Modifications: None

Vehicle ID: 495
Vehicle Name: Sandking
Modifications: None

Vehicle ID: 496
Vehicle Name: Blista Compact
Modifications: Transfender

Vehicle ID: 498
Vehicle Name: Boxville
Modifications: None

Vehicle ID: 499
Vehicle Name: Benson
Modifications: None

Vehicle ID: 500
Vehicle Name: Mesa
Modifications: Transfender

Vehicle ID: 502
Vehicle Name: Hotring Racer
Modifications: None

Vehicle ID: 503
Vehicle Name: Hotring Racer
Modifications: None

Vehicle ID: 504
Vehicle Name: Bloodring Banger
Modifications: None

Vehicle ID: 505
Vehicle Name: Rancher
Modifications: Transfender

Vehicle ID: 506
Vehicle Name: Super GT
Modifications: Transfender

Vehicle ID: 507
Vehicle Name: Elegant
Modifications: Transfender

Vehicle ID: 508
Vehicle Name: Journey
Modifications: None

Vehicle ID: 514
Vehicle Name: Tanker
Modifications: None

Vehicle ID: 515
Vehicle Name: Roadtrain
Modifications: None

Vehicle ID: 516
Vehicle Name: Nebula
Modifications: Transfender

Vehicle ID: 517
Vehicle Name: Majestic
Modifications: Transfender

Vehicle ID: 518
Vehicle Name: Buccaneer
Modifications: Transfender

Vehicle ID: 524
Vehicle Name: Cement Truck
Modifications: None

Vehicle ID: 525
Vehicle Name: Towtruck
Modifications: None

Vehicle ID: 526
Vehicle Name: Fortune
Modifications: Transfender

Vehicle ID: 527
Vehicle Name: Cadrona
Modifications: Transfender

Vehicle ID: 528
Vehicle Name: FBI Truck
Modifications: None

Vehicle ID: 529
Vehicle Name: Willard
Modifications: Transfender

Vehicle ID: 530
Vehicle Name: Forklift
Modifications: None

Vehicle ID: 531
Vehicle Name: Tractor
Modifications: None

Vehicle ID: 532
Vehicle Name: Combine Harvester
Modifications: None

Vehicle ID: 533
Vehicle Name: Feltzer
Modifications: Transfender

Vehicle ID: 534
Vehicle Name: Remington
Modifications: Loco Low Co

Vehicle ID: 535
Vehicle Name: Slamvan
Modifications: Loco Low Co

Vehicle ID: 536
Vehicle Name: Blade
Modifications: Loco Low Co

Vehicle ID: 539
Vehicle Name: Vortex
Modifications: None

Vehicle ID: 540
Vehicle Name: Vincent
Modifications: Transfender

Vehicle ID: 541
Vehicle Name: Bullet
Modifications: Transfender

Vehicle ID: 542
Vehicle Name: Clover
Modifications: Transfender

Vehicle ID: 543
Vehicle Name: Sadler
Modifications: None

Vehicle ID: 544
Vehicle Name: Firetruck LA
Modifications: None

Vehicle ID: 545
Vehicle Name: Hustler
Modifications: Transfender

Vehicle ID: 546
Vehicle Name: Intruder
Modifications: Transfender

Vehicle ID: 547
Vehicle Name: Primo
Modifications: Transfender

Vehicle ID: 549
Vehicle Name: Tampa
Modifications: Transfender

Vehicle ID: 550
Vehicle Name: Sunrise
Modifications: Transfender

Vehicle ID: 551
Vehicle Name: Merit
Modifications: Transfender

Vehicle ID: 552
Vehicle Name: Utility Van
Modifications: None

Vehicle ID: 554
Vehicle Name: Yosemite
Modifications: None

Vehicle ID: 555
Vehicle Name: Windsor
Modifications: Transfender

Vehicle ID: 556
Vehicle Name: Monster "A"
Modifications: None

Vehicle ID: 557
Vehicle Name: Monster "B"
Modifications: None

Vehicle ID: 558
Vehicle Name: Uranus
Modifications: Wheel Arch Angels

Vehicle ID: 559
Vehicle Name: Jester
Modifications: Wheel Arch Angels

Vehicle ID: 560
Vehicle Name: Sultan
Modifications: Wheel Arch Angels

Vehicle ID: 561
Vehicle Name: Stratum
Modifications: Wheel Arch Angels

Vehicle ID: 562
Vehicle Name: Elegy
Modifications: Wheel Arch Angels

Vehicle ID: 565
Vehicle Name: Flash
Modifications: Wheel Arch Angels

Vehicle ID: 566
Vehicle Name: Tahoma
Modifications: Loco Low Co

Vehicle ID: 567
Vehicle Name: Savanna
Modifications: Loco Low Co

Vehicle ID: 568
Vehicle Name: Bandito
Modifications: None

Vehicle ID: 571
Vehicle Name: Kart
Modifications: None

Vehicle ID: 572
Vehicle Name: Mower
Modifications: None

Vehicle ID: 573
Vehicle Name: Dune
Modifications: None

Vehicle ID: 574
Vehicle Name: Sweeper
Modifications: None

Vehicle ID: 575
Vehicle Name: Broadway
Modifications: Transfender

Vehicle ID: 576
Vehicle Name: Tornado
Modifications: Loco Low Co

Vehicle ID: 578
Vehicle Name: DFT-30
Modifications: None

Vehicle ID: 579
Vehicle Name: Huntley
Modifications: Transfender

Vehicle ID: 580
Vehicle Name: Stafford
Modifications: Transfender

Vehicle ID: 582
Vehicle Name: Newsvan
Modifications: None

Vehicle ID: 583
Vehicle Name: Tug
Modifications: None

Vehicle ID: 585
Vehicle Name: Emperor
Modifications: Transfender

Vehicle ID: 587
Vehicle Name: Euros
Modifications: Transfender

Vehicle ID: 588
Vehicle Name: Hotdog
Modifications: None

Vehicle ID: 589
Vehicle Name: Club
Modifications: Transfender

Vehicle ID: 596
Vehicle Name: Police Car (LSPD)
Modifications: None

Vehicle ID: 597
Vehicle Name: Police Car (SFPD)
Modifications: None

Vehicle ID: 598
Vehicle Name: Police Car (LVPD)
Modifications: None

Vehicle ID: 599
Vehicle Name: Police Ranger
Modifications None**:**

Vehicle ID: 600
Vehicle Name: Picador
Modifications: Transfender

Vehicle ID: 601
Vehicle Name: S.W.A.T.
Modifications: None

Vehicle ID: 602
Vehicle Name: Alpha
Modifications: Transfender

Vehicle ID: 603
Vehicle Name: Phoenix
Modifications: Transfender

Vehicle ID: 604
Vehicle Name: Glendale Shit
Modifications: None

Vehicle ID: 605
Vehicle Name: Sadler Shit
Modifications: None

Vehicle ID: 609
Vehicle Name: Boxville
Modifications: None

Поезда и вагоны

Vehicle ID: 537
Vehicle Name: Freight (Train)
Modifications: None

Vehicle ID: 538
Vehicle Name: Brownstreak (Train)
Modifications: None

Vehicle ID: 569
Vehicle Name: Freight Flat Trailer (Train)
Modifications: None

Vehicle ID: 570
Vehicle Name: Streak Trailer (Train)
Modifications: None

Vehicle ID: 590
Vehicle Name: Freight Box Trailer (Train)
Modifications: None

Велосипеды

Vehicle ID: 481
Vehicle Name: BMX
Modifications: None

Vehicle ID: 509
Vehicle Name: Bike
Modifications: None

Vehicle ID: 510
Vehicle Name: Mountain Bike
Modifications: None

Прицепы

Vehicle ID: 435
Vehicle Name: Article Trailer
Modifications: None

Vehicle ID: 450
Vehicle Name: Article Trailer 2
Modifications: None

Vehicle ID: 584
Vehicle Name: Petrol Trailer
Modifications: None

Vehicle ID: 591
Vehicle Name: Article Trailer 3
Modifications: None

Vehicle ID: 606
Vehicle Name: Baggage Trailer "A"
Modifications: None

Vehicle ID: 607
Vehicle Name: Baggage Trailer "B"
Modifications: None

Vehicle ID: 608
Vehicle Name: Tug Stairs Trailer
Modifications: None

Vehicle ID: 610
Vehicle Name: Farm Trailer
Modifications: None

Vehicle ID: 611
Vehicle Name: Utility Trailer
Modifications: None

Мотоциклы, мопеды и байки

Vehicle ID: 448
Vehicle Name: Pizzaboy
Modifications: None

Vehicle ID: 461
Vehicle Name: PCJ-600
Modifications: None

Vehicle ID: 462
Vehicle Name: Faggio
Modifications: None

Vehicle ID: 463
Vehicle Name: Freeway
Modifications: None

Vehicle ID: 468
Vehicle Name: Sanchez
Modifications: None

Vehicle ID: 471
Vehicle Name: Quad
Modifications: None

Vehicle ID: 521
Vehicle Name: FCR-900
Modifications: None

Vehicle ID: 522
Vehicle Name: NRG-500
Modifications: None

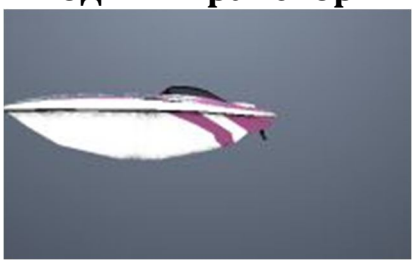
Vehicle ID: 523
Vehicle Name: HPV1000
Modifications: None

Vehicle ID: 581
Vehicle Name: BF-400
Modifications: None

Vehicle ID: 586
Vehicle Name: Wayfarer
Modifications: None

Водный транспорт

Vehicle ID: 430
Vehicle Name: Predator
Modifications: None



Vehicle ID: 446
Vehicle Name: Squallo
Modifications: None

Vehicle ID: 452
Vehicle Name: Speeder
Modifications: None

Vehicle ID: 453
Vehicle Name: Reefer
Modifications: None

Vehicle ID: 454
Vehicle Name: Tropic
Modifications: None

Vehicle ID: 472
Vehicle Name: Coastguard
Modifications: None

Vehicle ID: 473
Vehicle Name: Dinghy
Modifications: None

Vehicle ID: 484
Vehicle Name: Marquis
Modifications: None

Vehicle ID: 493
Vehicle Name: Jetmax
Modifications: None

Vehicle ID: 595
Vehicle Name: Launch
Modifications: None

Воздушный транспорт

Vehicle ID: 417
Vehicle Name: Leviathan
Modifications: None

Vehicle ID: 425
Vehicle Name: Hunter
Modifications: None

Vehicle ID: 447
Vehicle Name: Seasparrow
Modifications: None

Vehicle ID: 460
Vehicle Name: Skimmer
Modifications: None

Vehicle ID: 469
Vehicle Name: Sparrow
Modifications: None

Vehicle ID: 476
Vehicle Name: Rustler
Modifications: None

Vehicle ID: 487
Vehicle Name: Maverick
Modifications: None

Vehicle ID: 488
Vehicle Name: SAN News Maverick
Modifications: None

Vehicle ID: 497
Vehicle Name: Police Maverick
Modifications: None

Vehicle ID: 511
Vehicle Name: Beagle
Modifications: None

Vehicle ID: 512
Vehicle Name: Cropduster
Modifications: None

Vehicle ID: 513
Vehicle Name: Stuntplane
Modifications: None

Vehicle ID: 519
Vehicle Name: Shamal
Modifications: None

Vehicle ID: 520
Vehicle Name: Hydra
Modifications: None

Vehicle ID: 548
Vehicle Name: Cargobob
Modifications: None

Vehicle ID: 553
Vehicle Name: Nevada
Modifications: None

Vehicle ID: 563
Vehicle Name: Raindance
Modifications: None

Vehicle ID: 577
Vehicle Name: AT400
Modifications: None

Vehicle ID: 592
Vehicle Name: Andromada
Modifications: None

Vehicle ID: 593
Vehicle Name: Dodo
Modifications: None

RC транспорт

Vehicle ID: 441
Vehicle Name: RC Bandit
Modifications: None

Vehicle ID: 464
Vehicle Name: RC Baron
Modifications: None

Vehicle ID: 465
Vehicle Name: RC Raider
Modifications: None

Vehicle ID: 501
Vehicle Name: RC Goblin
Modifications: None

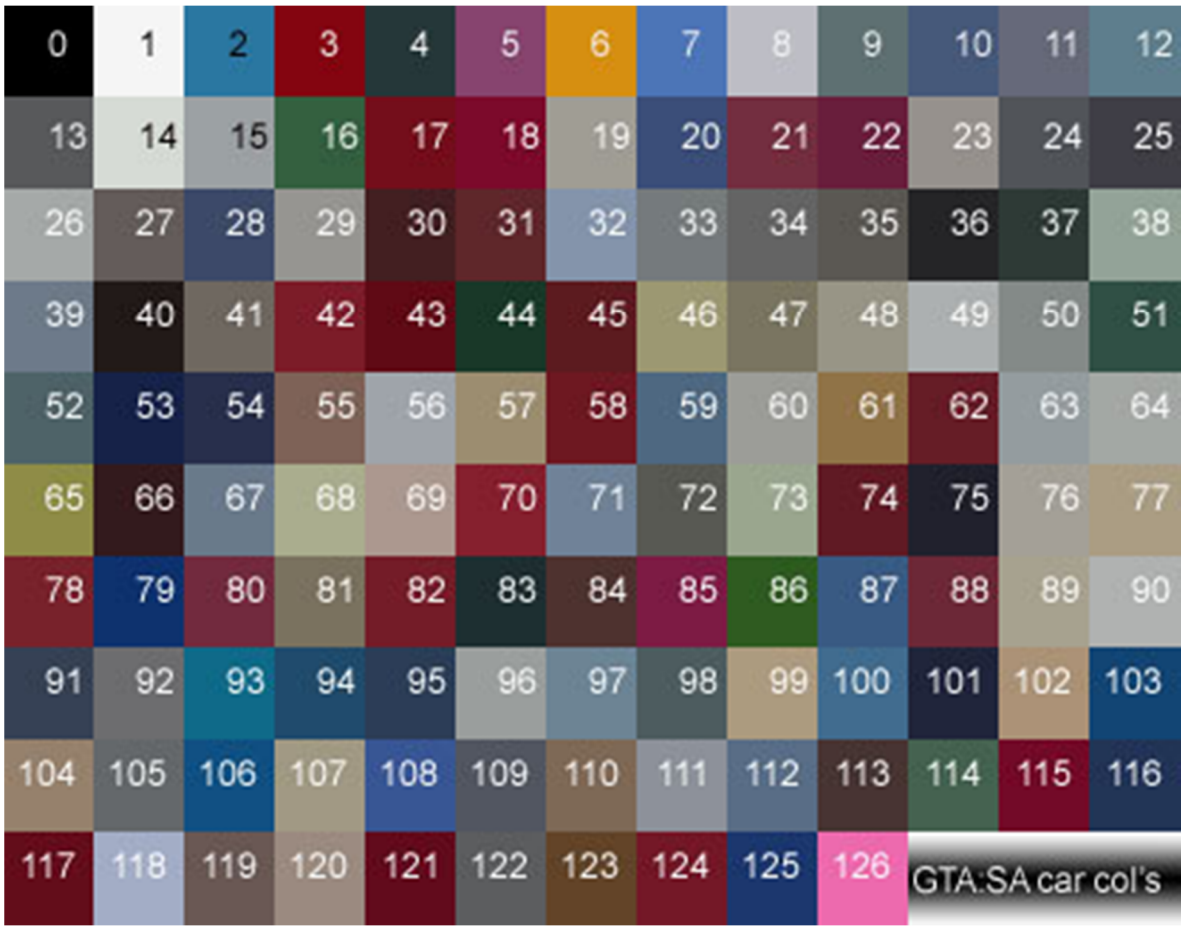
Vehicle ID: 564
Vehicle Name: RC Tiger
Modifications: None

Vehicle ID: 594
Vehicle Name: RC Cam
Modifications: None

ID цветов и их HEX коды

ID цветов для транспорта

CarColorIDs - между 0 и 126. Эти ID цветов вы можете использовать для автомобилей:



HEX-коды цветов

HEX-коды применяются для отправки цветных сообщений в чат и обычно используются в функциях:

- **SendClientMessage**
- **SendClientMessageToAll**
- **SendPlayerMessageToPlayer**
- **SendPlayerMessageToAll**
- **SetPlayerColor** (и еще в некоторых функциях)

Таблица «безопасных» цветов

255.255.204 FFFFCC	255.255.153 FFFF99	255.255.102 FFFF66	255.255.51 FFFF33	255.255.0 FFFF00	204.204.0 CCCC00
255.204.102 FFCC66	255.204.0 FFCC00	255.204.51 FFCC33	204.153.0 CC9900	204.153.51 CC9933	153.102.0 996600
255.153.0 FF9900	255.153.51 FF9933	204.153.102 CC9966	204.102.0 CC6600	153.102.51 996633	102.51.0 663300

255.204.153 FFCC99	255.153.102 FF9966	255.102.0 FF6600	204.102.51 CC6633	153.51.0 993300	102.0.0 660000
255.102.51 FF6633	204.51.0 CC3300	255.51.0 FF3300	255.0.0 FF0000	204.0.0 CC0000	153.0.0 990000
255.204.204 FFCCCC	255.153.153 FF9999	255.102.102 FF6666	255.51.51 FF3333	255.0.51 FF0033	204.0.51 CC0033
204.153.153 CC9999	204.102.102 CC6666	204.51.51 CC3333	153.51.51 993333	153.0.51 990033	51.0.0 330000
255.102.153 FF6699	255.51.102 FF3366	255.0.102 FF0066	204.51.102 CC3366	153.102.102 996666	102.51.51 663333
255.153.204 FF99CC	255.51.153 FF3399	255.0.153 FF0099	204.0.102 CC0066	153.51.102 993366	102.0.51 660033
255.102.204 FF66CC	255.0.204 FF00CC	255.51.204 FF33CC	204.102.153 CC6699	204.0.153 CC0099	153.0.102 990066
255.204.255 FFCCFF	255.153.255 FF99FF	255.102.255 FF66FF	255.51.255 FF33FF	255.0.255 FF00FF	204.51.153 CC3399
204.153.204 CC99CC	204.102.204 CC66CC	204.0.204 CC00CC	204.51.204 CC33CC	153.0.153 990099	153.51.153 993399
204.102.255 CC66FF	204.51.255 CC33FF	204.0.255 CC00FF	153.0.204 9900CC	153.102.153 996699	102.0.102 660066
204.153.255 CC99FF	153.51.204 9933CC	153.51.255 9933FF	153.0.255 9900FF	102.0.153 660099	102.51.102 663366
153.102.204 9966CC	153.102.255 9966FF	102.0.204 6600CC	102.51.204 6633CC	102.51.153 663399	51.0.51 330033
204.204.255 CCCCFF	153.153.255 9999FF	102.51.255 6633FF	102.0.255 6600FF	51.0.153 330099	51.0.102 330066
153.153.204 9999CC	102.102.255 6666FF	102.102.204 6666CC	102.102.153 666699	51.51.153 333399	51.51.102 333366
51.51.255 3333FF	51.0.255 3300FF	51.0.204 3300CC	51.51.204 3333CC	0.0.153 000099	0.0.102 000066
102.153.255 6699FF	51.102.255 3366FF	0.0.255 0000FF	0.0.204 0000CC	0.51.204 0033CC	0.0.51 000033
0.102.255 0066FF	0.102.204 0066CC	51.102.204 3366CC	0.51.255 0033FF	0.51.153 003399	0.51.102 003366

153.204.255 99CCFF	51.153.255 3399FF	0.153.255 0099FF	102.153.204 6699CC	51.102.153 336699	0.102.153 006699
102.204.255 66CCFF	51.204.255 33CCFF	0.204.255 00CCFF	51.153.204 3399CC	0.153.204 0099CC	0.51.51 003333
153.204.204 99CCCC	102.204.204 66CCCC	51.153.153 339999	102.153.153 669999	0.102.102 006666	51.102.102 336666
204.255.255 CCFFFF	153.255.255 99FFFF	102.255.255 66FFFF	51.255.255 33FFFF	0.255.255 00FFFF	0.204.204 00CCCC
153.255.204 99FFCC	102.255.204 66FFCC	51.255.204 33FFCC	0.255.204 00FFCC	51.204.204 33CCCC	0.153.153 009999
102.204.153 66CC99	51.204.153 33CC99	0.204.153 00CC99	51.153.102 339966	0.153.102 009966	0.102.51 006633
102.255.153 66FF99	51.255.153 33FF99	0.255.153 00FF99	51.204.102 33CC66	0.204.102 00CC66	0.153.51 009933
153.255.153 99FF99	102.255.102 66FF66	51.255.102 33FF66	0.255.102 00FF66	51.153.51 339933	0.102.0 006600
204.255.204 CCFFCC	153.204.153 99CC99	102.204.102 66CC66	102.153.102 669966	51.102.51 336633	0.51.0 003300
51.255.51 33FF33	0.255.51 00FF33	0.255.0 00FF00	0.204.0 00CC00	51.204.51 33CC33	0.204.51 00CC33
102.255.0 66FF00	102.255.51 66FF33	51.255.0 33FF00	51.204.0 33CC00	51.153.0 339900	0.153.0 009900
204.255.153 CCFF99	153.255.102 99FF66	102.204.0 66CC00	102.204.51 66CC33	102.153.51 669933	51.102.0 336600
153.255.0 99FF00	153.255.51 99FF33	153.204.102 99CC66	153.204.0 99CC00	153.204.51 99CC33	102.153.0 669900
204.255.102 CCFF66	204.255.0 CCFF00	204.255.51 CCFF33	204.204.153 CCCC99	102.102.51 666633	51.51.0 333300
204.204.102 CCCC66	204.204.51 CCCC33	153.153.51 999966	153.153.102 999933	153.153.0 999900	102.102.0 666600
255.255.255 FFFFFF	204.204.204 CCCCCC	153.153.153 999999	102.102.102 666666	51.51.51 333333	0.0.0 000000

IV. Описание функций Pawn

В этом разделе учебника будут приведены описания функций, их структуры и небольшие примеры их использования как для стандартных инклудов, так и для сторонних.

Функции A_SAMP.INC

Основные функции

Print

Функция посылает текст в консоль сервера (все сообщения в консоли сохраняются в server_log).

Синтаксис

print(const string[]);

Параметры функции

const string[] – текст который нужно вывести в консоль.

Возможное использование:

```
1  print("Blank Gamemode by your name here");
```

Printf

Функция посылает текст в консоль сервера (все сообщения в консоли сохраняются в server_log).

Синтаксис

printf(const format[], {Float,_}:...);

Параметры функции

const string[] – строка которую нужно вывести в консоль.
{Float,_}:... – текст или переменная, которую нужно передать в строку

Возможное использование:

```
1  printf("%s Gamemode by your name here",”Blank”);
```

Format

Функция форматирует строку передавая в нее значение других переменных или строк.

Синтаксис

format(output[], len, const format[], {Float,_}:...);

Параметры функции

output[] – переменная в которой присваивается отформатированная строка
len – размер переменной для присвоения отформатированной строки.
const format [] – текст который нужно форматировать
{Float,_}:... – текст или переменная, которую нужно передать в строку

Возможное использование:

```
1  new number = 42;
2  printf("The number is %d.",number); //The number is 42
```

SendClientMessage

Эта функция используется для отправки сообщения игроку в окно чата.

Синтаксис

SendClientMessage(playerid, color, const message[]);

Параметры функции

playerid – ID игрока, которому отправляется сообщение в чат.
color – HEX-код цвета, которым будет выделено сообщение.
const message[] – текст сообщения, которое вы хотите отправить.

Возможное использование:

```
1  SendClientMessage(playerid, 0xFFFF00AA,"Это сообщение выделено желтым цветом");
```

SendClientMessageToAll

Эта функция используется для отправки сообщения всем игрокам в окно чата.

Синтаксис

SendClientMessageToAll(color, const message[]);

Параметры функции

color – HEX-код цвета, которым будет выделено сообщение.
const message[] – текст сообщения, которое вы хотите отправить.

Возможное использование:

```
1  SendClientMessage(0xFFFF00AA,"Это сообщение выделено желтым цветом");
```

SendPlayerMessageToPlayer

Эта функция используется для отправки сообщения от имени игрока другому игроку в окно чата.

Синтаксис

SendPlayerMessageToPlayer(playerid, senderid, const message[]);

Параметры функции

playerid – ID игрока, которому будет отправлено сообщение
senderid – ID игрока, от которого будет отправлено сообщение
const message[] – текст сообщения, который будет отправлен.

Возможное использование:

```
1  SetPlayerMessageToPlayer(playerid,senderid,"Привет!");
```

SendPlayerMessageToAll

Эта функция используется для отправки сообщения от имени игрока всем игрокам в окно чата.

Синтаксис

SendPlayerMessageToAll(senderid, const message[]);

Параметры функции

senderid – ID игрока, от которого будет отправлено сообщение
const message[] – текст сообщения, который будет отправлен.

Возможное использование:

```
1  SetPlayerMessageToAll(senderid,"Привет!");
```

SendDeathMessage

Отправляет сообщение о смерти игрока на сервер, которое отобразится в правой нижней части экрана у игрока под HUD`ом.

Синтаксис

SendDeathMessage(killer,killee,weapon);

Параметры функции

killer – ID игрока который убил другого игрока
killed – ID игрока, которого убили
weapon – Причина смерти игрока

Возможное использование:

```
1  SendDeathMessage(killerid,playerid,reason);
```

GameTextForAll

Эта функция отображает большой текст на экране у каждого игрока.

Синтаксис

GameTextForAll(const string[],time,style);

Параметры функции

const string[] – Текст, который нужно вывести на экран
time – Время в течении которого будет показан текст, в миллисекундах
style – Стиль отображаемоего текста (смотрите «Стили текста.doc» прилагается к учебнику).

Возможное использование:

```
1  GameTextForAll("Добро пожаловать на сервер",5000, 3); //Для функции OnPlayerConnect
```

GameTextForPlayer

Эта функция отображает большой текст на экране для определенного игрока.

Синтаксис

GameTextForPlayer(playerid,const string[],time,style);

Параметры функции

playerid – ID игрока, у которого отобразится этот текст
const string[] – Текст, который нужно вывести на экран
time – Время в течении которого будет показан текст, в миллисекундах
style – Стиль отображаемоего текста (смотрите «Стили текста.doc» прилагается к учебнику).

Возможное использование:

```
1  GameTextForPlayer(playerid,"Добро пожаловать на сервер",5000, 3); //Для функции OnPlayerConnect
```

SetTimer

Устанавливает таймер, на выполнение какой-либо функции.

Синтаксис

SetTimer(funcname[], interval, repeating);

Параметры функции

functionname[] – Название public функции, которая будет вызываться таймером

interval – Время (в миллисекундах.)
repeating – Определяет, будет ли таймер повторяться или нет, 1 – да, 0 – нет.

Возможное использование:

```
1  SetTimer("CountDown",1000,1);
```

SetTimerEx

Устанавливает таймер, на выполнение какой-либо функции с параметрами.

Синтаксис

SetTimerEx(funcname[], interval, repeating, const format[], {Float,_}:...);

Параметры функции

functionname[] – Название public функции, которая будет вызываться таймером
interval – Время (в миллисекундах.)
repeating – Определяет, будет ли таймер повторяться или нет, 1 – да, 0 – нет.
const format[] – текст строки
{Float,_}:... - текст или переменная, которую нужно передать в строку

Возможное использование:

```
1  new timer = SetTimer("PutPlayer",1000,0,"df",playerid,500.0);
```

KillTimer

Удаляем таймер с заданным ID.

Синтаксис

SetTimerEx(funcname[], interval, repeating, const format[], {Float,_}:...);

Параметры функции

functionname[] – Название public функции, которая будет вызываться таймером
interval – Время (в миллисекундах.)
repeating – Определяет, будет ли таймер повторяться или нет, 1 – да, 0 – нет.
const format[] – текст строки
{Float,_}:... - текст или переменная, которую нужно передать в строку

Возможное использование:

```
1  new timer = SetTimer("CountDown",1000,1); //Создаем таймер и передаем его ID в переменную times
2  KillTimer(timer); //Удаляем таймер
```

GetTickCount

Получает число процессорных тиков со времени последнего перезапуска сервера. Эта функция работает только в Windows!

Синтаксис

GetTickCount();

Параметры функции

нет параметров

Возможное использование:

```
1  new Ticks = GetTickCount();
```

GetMaxPlayers

Эта функция возвращает максимальное число игроков, которое может играть на Вашем сервере.

Синтаксис

GetMaxPlayers();

Параметры функции

нет параметров

Возможное использование:

```
1  new Players = GetMaxPlayers();
```

CallLocalFunction

Позволяет вызвать любую функцию из данного игрового режима или фильтр скрипта.

Синтаксис

CallLocalFunction(const function[], const format[], {Float,_}:...);

Параметры функции

const function[] – Имя функции
conts format[] – Строка для форматирования параметров
{Float,_}:... – Параметры

Возможное использование:

```
1  CallLocalFunction("IsPlayerOnBike","i",playerid);
```

CallRemoteFunction

Позволяет вызвать любую функцию из любого запущенного игрового режима и фильтр скрипта.

Синтаксис

CallRemoteFunction(const function[], const format[], {Float,_}:...);

Параметры функции

const function[] – Имя функции
conts format[] – Строка для форматирования параметров
{Float,_}:... – Параметры

Возможное использование:

```
1  CallRemoteFunction("IsPlayerOnBike","i",playerid);
```

SetGameModeText

Устанавливает имя режиме игры, которое отображается в списке серверов клиента.

Синтаксис

SetGameModeText(const string[]);

Параметры функции

const string[] – Имя режима игры

Возможное использование:

```
1  SetGameModeText("Las Venturas Team Death Match");
```

AddPlayerClass

Эта функция добавляет новый класс в режим игры. Эта функция должна вызываться только из автовызываемой функции **OnGameModeInit**, в другом месте она не будет работать.

Синтаксис

AddPlayerClass(modelid, Float:spawn_x, Float:spawn_y, Float:spawn_z, Float:z_angle, weapon1, weapon1_ammo, weapon2, weapon2_ammo, weapon3, weapon3_ammo);

Параметры функции

modelid – ID скина;
Float:spawn_x – координата-X стартовой позиции игрока;
Float:spawn_y – координата-Y стартовой позиции игрока;
Float:spawn_z – координата-Z стартовой позиции игрока;
Float:z_angle – Угол поворота игрока после размещения в стартовой позиции
weapon1 – ID первого оружия игрока.
weapon1_ammo – Количество патронов для первого оружия игрока
weapon2 – ID второго оружия игрока.
weapon2_ammo – Количество патронов для второго оружия игрока
weapon3 – ID третья оружия игрока.
weapon3_ammo – Количество патронов для третьего оружия игрока

Возможное использование:

```
1  AddPlayerClass(270, 1958.3783, 1343.1572, 15.3746, 269.1425, 0, 0, 0, 0, 0, 0);
```

AddPlayerClassEx

Эта функция добавляет новый класс в режим игры и предоставляет возможность выбрать игроку команду перед его размещением в стартовой позиции. Эта функция должна вызываться только из автовызываемой функции **OnGameModeInit**, в другом месте она не будет работать.

Синтаксис

AddPlayerClassEx(teamid, modelid, Float:spawn_x, Float:spawn_y, Float:spawn_z, Float:z_angle, weapon1, weapon1_ammo, weapon2, weapon2_ammo, weapon3, weapon3_ammo);

Параметры функции

teamid – команда к которой принадлежит данный класс
modelid – ID скина;
Float:spawn_x – координата-X стартовой позиции игрока;
Float:spawn_y – координата-Y стартовой позиции игрока;
Float:spawn_z – координата-Z стартовой позиции игрока;
Float:z_angle – Угол поворота игрока после размещения в стартовой позиции
weapon1 – ID первого оружия игрока.
weapon1_ammo – Количество патронов для первого оружия игрока
weapon2 – ID второго оружия игрока.
weapon2_ammo – Количество патронов для второго оружия игрока
weapon3 – ID третья оружия игрока.
weapon3_ammo – Количество патронов для третьего оружия игрока

Возможное использование:

```
1  AddPlayerClass(270, 1958.3783, 1343.1572, 15.3746, 269.1425, 0, 0, 0, 0, 0, 0);
```

SetTeamCount

Устанавливает имя режиме игры, которое отображается в списке серверов клиента.

Синтаксис

SetTeamCount(count);

Параметры функции

count – число команд

Возможное использование:

```
1  SetTeamCount(5);
```

AddStaticVehicle

Эта функция размещает транспорт в вашем режиме игры. Эта функция должна вызываться только из автовызываемой функции OnGameModeInit, в другом месте она не будет работать.

Синтаксис

AddStaticVehicle(modelid, Float:spawn_x, Float:spawn_y, Float:spawn_z, Float:z_angle, color1, color2);

Параметры функции

modelid – ID транспорта
Float:spawn_x – Координата X для стартовой позиции транспорта
Float:spawn_y – Координата Y для стартовой позиции транспорта
Float:spawn_z – Координата Z для стартовой позиции транспорта
Float:z_angle – Угол поворота транспорта на стартовой позиции
color1 – ID первичного цвета транспорта
color2 – ID вторичного цвета транспорта

Возможное использование:

```
1  AddStaticVehicle(401,2822.0801, 2169.3892, 10.5975, 270.8597, 59, 59); //Bravura
```

AddStaticVehicleEx

Эта функция размещает транспорт в вашем режиме игры. Эта функция должна вызываться только из автовызываемой функции OnGameModeInit, в другом месте она не будет работать.

Синтаксис

AddStaticVehicleEx(modelid, Float:spawn_x, Float:spawn_y, Float:spawn_z, Float:z_angle, color1, color2, respawn_delay);

Параметры функции

modelid – ID транспорта
Float:spawn_x – Координата X для стартовой позиции транспорта
Float:spawn_y – Координата Y для стартовой позиции транспорта
Float:spawn_z – Координата Z для стартовой позиции транспорта
Float:z_angle – Угол поворота транспорта на стартовой позиции
color1 – ID первичного цвета транспорта
color2 – ID вторичного цвета транспорта
respawn_delay – задержка перед переразмещением транспорта в стартовой позиции.

Возможное использование:

```
1  AddStaticVehicle(401,2822.0801, 2169.3892, 10.5975, 270.8597, 59, 59, 600); //Bravura
```

AddStaticPickup

Эта функция добавляет статический пикап - вещь, которую можно подобрать: здоровье, броня, оружие (Используйте ID МОДЕЛИ оружия, а НЕ ID оружия!) и объекты.

Синтаксис

AddStaticPickup(model, type, Float:X, Float:Y, Float:Z, virtualworld = 0);

Параметры функции

model – ID модели для подбираемой пикапа
type – Тип появления вещи
Float:X – Координата X для стартовой позиции пикапа
Float:Y – Координата Y для стартовой позиции пикапа
Float:Z – Координата Z для стартовой позиции пикапа

Возможное использование:

```
1  AddStaticPickup(1212, 15,2822.0801, 2169.3892, 10.5975); //Деньги
```

Доступные типы пикапов:

0 – Пикап не отображается.
2 – Берётся, появляется после некоторого времени.
3 – Берется, но не появляется.
4 – Исчезает вскоре после появления (применяется для выброшенного оружия)
5 – Исчезает вскоре после появления (применяется для выброшенного оружия)
8 – Берётся, но не имеет эффекта. Исчезает автоматически.
11 – Взрывается через несколько секунд после того как появится (бомба)
12 – Взрывается через несколько секунд после того как появится.
13 – Медленно decends на землю.
15 – Берется, но не появляется.
19 – Берётся, но не имеет эффекта (информационные иконки)
22 – Берется, но не появляется
23 – Берётся, но не исчезает.

DestroyPickup

Убирает с карты (уничтожает) пикап с заданным ID.

Синтаксис

DestroyPickup(pickup);

Параметры функции

pickup – ID пикапа, который нужно убрать

Возможное использование:

```
1  new PickMoney = AddStaticPickup(1212, 15,2822.0801, 2169.3892, 10.5975); //Деньги
2  DestroyPickup(PickMoney); //удаление пикапа
```

ShowNameTags

Показать или скрыть таги с именами игроков

Синтаксис

ShowNameTags(show);

Параметры функции

show – 1 показать, 0 – скрыть (по умолчанию – 1)

Возможное использование:

```
1  ShowNameTags(0);
```

ShowPlayerMarkers

Показать или скрыть маркеры игроков на радаре

Синтаксис

ShowPlayerMarkers(mode);

Параметры функции

mode – 1 показать, 0 – скрыть (по умолчанию – 1)

Возможное использование:

```
1  ShowPlayerMarkers(0);
```

SetWorldTime

Устанавливает игровое время на определенный час

Синтаксис

SetWorldTime(hour);

Параметры функции

hour – час.

Возможное использование:

```
1  SetWorldTime(12);
```

GetWeaponName

Функция возвращает имя оружия.

Синтаксис

GetWeaponName(weaponid, const weapon[], len);

Параметры функции

weaponid – ID оружия
const weapon[] – строка в которую будет передано имя оружия
len – размер строки

Возможное использование:

```
1  new weapon[20];
2  GetPlayerWeapon(35,weapon,sizeof(weapon));
```

AllowInteriorWeapons

Разрешает или запрещает использование оружие в интерьерах

Синтаксис

AllowInteriorWeapons(allow);

Параметры функции

allow – 1 разрешить, 0 – запретить (по умолчанию «разрешить» - 1)

Возможное использование:

```
1  GetPlayerWeapon(35,weapon,sizeof(weapon));
```

SetWeather

Устанавливает погоду на сервере

Синтаксис

SetWeather(weatherid);

Параметры функции

weatherid – ID погоды

Возможное использование:

```
1 SetWeather(35);
```

SetGravity

Устанавливает гравитацию на сервере

Синтаксис

SetGravity(Float:gravity);

Параметры функции

Float:gravity – значение гравитации

Возможное использование:

```
1 SetGravity(0,008);
```

AllowAdminTeleport

Разрешает или запрещает администратору сервера телепортироваться меткой на карте.

Синтаксис

AllowAdminTeleport(allow);

Параметры функции

allow – 1 разрешить, 0 – запретить (по умолчанию «запретить» - 0)

Возможное использование:

```
1 AllowAdminTeleport(1);
```

SetDeathDropAmount

Устанавливает количество денег выпадающих у игрока.

Синтаксис

SetDeathDropAmount(amount);

Параметры функции

amount – количество денег.

Возможное использование:

```
1 SetDeathDropAmount (1000);
```

CreateExplosion

Создает взрыв в указанном месте

Синтаксис

CreateExplosion(Float:X, Float:Y, Float:Z, type, Float:Radius);

Параметры функции

Float:X – Координата X для места взрыва
Float:Y – Координата Y для места взрыва
Float:Z – Координата Z для места взрыва
type – Тип взрыва
Float:Radius – Радиус взрыва

Возможное использование:

```
1 CreateExplosion(2822.0801, 2169.3892, 10.5975,3.0);
```

UsePlayerPedAnims

Делает стандартную анимацию у всех игроков как у CJ

Синтаксис

UsePlayerPedAnims();

Параметры функции

Нет параметров

Возможное использование:

```
1 UsePlayerPedAnims();
```

DisableInteriorEnterExits

Убирает все маркеры входов в здания

Синтаксис

DisableInteriorEnterExits();

Параметры функции

Нет параметров

Возможное использование:

```
1 DisableInteriorEnterExits();
```

SetNameTagDrawDistance

Устанавливает дистанцию видимости тагов игроков

Синтаксис

SetNameTagDrawDistance(Float:distance);

Параметры функции

Float:distance – дистанция видимости

Возможное использование:

```
1 SetNameTagDrawDistance(5,0);
```

DisableNameTagLOS

Устанавливает дистанцию видимости тагов игроков

Синтаксис

DisableNameTagLOS();

Параметры функции

Нет параметров

Возможное использование:

```
1  DisableNameTagLOS();
```

LimitGlobalChatRadius

Устанавливает дистанцию видимости сообщений чата

Синтаксис

LimitGlobalChatRadius(Float:chat_radius);

Параметры функции

Float:chat_radius – дистанция видимости сообщений чата

Возможное использование:

```
1  LimitGlobalChatRadius(5,0);
```

LimitPlayerMarkerRadius

Устанавливает дистанцию видимости маркера игрока на радаре

Синтаксис

LimitPlayerMarkerRadius(Float:marker_radius);

Параметры функции

Float:marker_radius – дистанция видимости маркера игрока на радаре

Возможное использование:

```
1  LimitPlayerMarkerRadius(5,0);
```

IsPlayerAdmin

Определяет, вошел ли игрок как RCON-администратор

Синтаксис

IsPlayerAdmin(playerid);

Параметры функции

playerid – ID проверяемого игрока

Возможное использование:

```
1  if(IsPlayerAdmin(playerid))
```

Kick

Кикнуть игрока

Синтаксис

Kick(playerid);

Параметры функции

playerid – ID игрока

Возможное использование:

```
1  Kick(playerid);
```

Ban

Забанить игрока

Синтаксис

Ban(playerid);

Параметры функции

playerid – ID игрока

Возможное использование:

```
1  Ban(playerid);
```

BanEx

Банит игрока на вашем сервере и записывает причину бана
в файл samp.ban. Игрок будет забанен на основе IP-адреса.

Синтаксис

BanEx(playerid, const reason[]);

Параметры функции

playerid – ID игрока
const reason[] – причина бана

Возможное использование:

```
1  BanEx(playerid,reason);
```

Функции для работы с меню

CreateMenu

Создает меню в памяти сервера

Синтаксис

CreateMenu(const title[], columns, Float:x, Float:y, Float:col1width, Float:col2width = 0.0);

Параметры функции

show – 1 – показывать, 0 – скрывать подсвечивание имен.
title[] – Заголовок
columns – Кол-во столбцов (1/2)
Float:x – Координата X на экране при разрешении 640x480
Float:y – Координата Y
Float:col1width – Ширина 1-го столбца
Float:col2width – Ширина 2-го столбца

Возможное использование:

```
1  new Menu:examplemenu;
2  examplemenu = CreateMenu("Your Menu", 2, 200.0, 100.0, 150.0, 150.0);
```

DestroyMenu

Стирает созданное меню из памяти сервера.

Синтаксис

DestroyMenu(Menu:menuid);

Параметры функции

Menu:menuid – ID меню

Возможное использование:

```
1  new Menu:examplemenu;
2  examplemenu = CreateMenu("Your Menu", 2, 200.0, 100.0, 150.0, 150.0);
3  DestroyMenu (examplemenu);
```

AddMenuItem

Добавляет новый пункт в меню.

Синтаксис

AddMenuItem(Menu:menuid, column, const menutext[]);

Параметры функции

Menu:menuid – ID меню, в которое нужно добавить пункт
column – Номер колонки меню, в который нужно добавить пункт
const menutext[] – Заголовок пункта меню

Возможное использование:

```
1  AddMenuItem(examplemenu,1,"Weapon");
```

SetMenuColumnHeader

Устанавливает заголовок колонки меню

Синтаксис

SetMenuColumnHeader(Menu:menuid, column, const columnheader[]);

Параметры функции

Menu:menuid – ID меню
column – Номер колонки меню (колонки нумеруются с 0, возможно создать только 2 колонки).
const menutext[] – Заголовок колонки

Возможное использование:

```
1  SetMenuColumnHeader(examplemenu, 1,"Weapon");
```

ShowMenuForPlayer

Показывает созданное меню игроку

Синтаксис

ShowMenuForPlayer(Menu:menuid, playerid);

Параметры функции

Menu:menuid – ID меню
playerid – ID игрока, которому нужно показать меню

Возможное использование:

```
1 ShowMenuForPlayer(examplemenu,playerid);
```

HideMenuForPlayer

Скрывает созданное меню от игрока

Синтаксис

HideMenuForPlayer(Menu:menuid, playerid);

Параметры функции

Menu:menuid – ID меню
playerid – ID игрока, у которого нужно скрыть меню

Возможное использование:

```
1 HideMenuForPlayer(examplemenu,playerid);
```

IsValidMenu

Проверяет, создано ли меню

Синтаксис

IsValidMenu(Menu:menuid);

Параметры функции

Menu:menuid – ID меню

Возможное использование:

```
1 IsValidMenu(examplemenu);
```

DisableMenu

Отключает меню

Синтаксис

DisableMenu(Menu:menuid);

Параметры функции

Menu:menuid – ID меню

Возможное использование:

```
1 DisableMenu(examplemenu);
```

DisableMenuRow

Отключает одну из строк меню

Синтаксис

DisableMenuRow(Menu:menuid, row);

Параметры функции

Menu:menuid – ID меню
row - Индекс строки (ряда), который нужно отключить (индекс начинается с 0).

Возможное использование:

```
1  DisableMenuRow(examplemenu);
```

GetPlayerMenu

Возвращает ID меню которое в текущий момент показано игроку

Синтаксис

GetPlayerMenu(playerid);

Параметры функции

playerid – ID игрока, у которого нужно скрыть меню

Возможное использование:

```
1  new playermenu = GetPlayerMenu(playerid);
```

Функции для работы с TextDraw

TextDrawCreate

Создает новый Text Draw в памяти сервера и возвращает ID
Text Draw, который идентифицирует этот текст, переменная типа Text.

Синтаксис

TextDrawCreate(Float:x, Float:y, text[]);

Параметры функции

Float:x - Координата x позиции на экране, где будет показан Text Draw. По умолчанию считается, что разрешение экрана 640x480, для другого разрешения текст растягивается пропорционально.
Float:y - Координата y позиции на экране, где будет показан Text Draw.
text[] – Текст сообщения

Возможное использование:

```
1  new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
```

TextDrawDestroy

Стирает из памяти сервера Text Draw с данным ID.

Синтаксис

TextDrawDestroy(Text:text);

Параметры функции

Text:text - ID Text Draw, который нужно стереть

Возможное использование:

```
1  new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
2  TextDrawDestroy(Welcome);
```

TextDrawLetterSize

Устанавливает ширину и высоту букв.

Синтаксис

TextDrawLetterSize(Text:text, Float:x, Float:y);

Параметры функции

Text:text - ID Text Draw, ширину и высоту букв которого нужно изменить.
Float:x – Ширина букв в пикселях
Float:y – Высота букв в пикселях

Возможное использование:

```
1  new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
2  TextDrawLetterSize(Welcome,20.0,30.0);
```

TextDrawTextSize

Устанавливает ширину и высоту рамки (при условии, что она используется).

Синтаксис

TextDrawTextSize(Text:text, Float:x, Float:y);

Параметры функции

Text:text - ID Text Draw, ширину и высоту рамки которого нужно изменить.
Float:x – Ширина букв в пикселях
Float:y – Высота букв в пикселях

Возможное использование:

```
1  new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
2  TextDrawTextSize(Welcome,20.0,30.0);
```

TextDrawAlignment

Устанавливает смещение текста в Text Draw.

Синтаксис

TextDrawAlignment(Text:text, alignment);

Параметры функции

Text:text - ID Text Draw, смешение текста которого нужно установить.
alignment – ID смешения: 0 или 1 – влево, 2 – центр, 3 – вправо.

Возможное использование:

```
1  new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
2  TextDrawAlignment(Welcome,2);
```

TextDrawColor

Устанавливает цвет текста в Text Draw.

Синтаксис

TextDrawColor(Text:text, color);

Параметры функции

Text:text - ID Text Draw, цвет текста которого нужно изменить.
color – цвет текста

Возможное использование:

```
1  new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
2  TextDrawColor(Welcome,0x000000FF);
```

TextDrawUseBox

Определяет использование рамки.

Синтаксис

TextDrawUseBox(Text:text, use);

Параметры функции

Text:text - ID Text Draw, для которого нужно определить использование рамки.
use – 1 использовать, 0 – не использовать

Возможное использование:

```
1  new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
2  TextDrawUserBox(Welcome,2);
```

TextDrawBoxColor

Устанавливает цвет рамки.

Синтаксис

TextDrawBoxColor(Text:text, color);

Параметры функции

Text:text - ID Text Draw, для которого нужно определить использование рамки.
color – цвет рамки

Возможное использование:

```
1  new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
2  TextDrawBoxColor(Welcome,0x000000FF);
```

TextDrawSetShadow

Устанавливает размер тени текста.

Синтаксис

TextDrawSetShadow(Text:text, size);

Параметры функции

Text:text - ID Text Draw, для которого нужно установить тень текста.
size – Размер тени

Возможное использование:

```
1  new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
2  TextDrawSetShadow(Welcome,1);
```

TextDrawSetOutline

Устанавливает размер обводки текста.

Синтаксис

TextDrawSetOutline(Text:text, size);

Параметры функции

Text:text - ID Text Draw, для которого нужно установить обводку текста.
size – Толщина обводки

Возможное использование:

```
new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
TextDrawSetOutline(Welcome,1);
```

TextDrawBackgroundColor

Регулировка цвета текста и области фона..

Синтаксис

TextDrawSetOutline(Text:text, size);

Параметры функции

Text:text - ID Text Draw, для которого нужно установить шрифт текста.
color – цвет. (Если TextDrawSetOutline используется размером > 0, контуры цвета будут совпадать с цветом использования)

Возможное использование:

```
new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
TextDrawBackgroundColor(Welcome, 0x000000FF);
```

TextDrawFont

Изменяет шрифт текста.

Синтаксис

TextDrawBackgroundColor(Text:text, color);

Параметры функции

Text:text - ID Text Draw, для которого нужно установить шрифт текста.
font – ID шрифта

Возможное использование:

```
new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
TextDrawFont(Welcome,0);
```

TextDrawSetProportional

Устанавливает интервал между символами.

Синтаксис

TextDrawSetProportional(Text:text, set);

Параметры функции

text - Текст.
set - Пропорции.

Возможное использование:

```
new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
TextDrawSetProportional(Welcome,1);
```

TextDrawShowForPlayer

Показывает Text Draw с данным ID определенному игроку.

Синтаксис

TextDrawShowForPlayer(playerid, Text:text);

Параметры функции

Text:text – ID Text Draw
playerid – ID игрока, которому нужно показать данный Text Draw

Возможное использование:

```
1  new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
2  TextDrawShowPlayer(playerid,Welcome);
```

TextDrawHideForPlayer

Скрывает Text Draw с данным ID от определенного игрока.

Синтаксис

TextDrawHideForPlayer (playerid, Text:text);

Параметры функции

Text:text – ID Text Draw
playerid – ID игрока, которому нужно показать данный Text Draw

Возможное использование:

```
1  new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
2  TextDrawHideForPlayer(playerid,Welcome);
```

TextDrawShowForAll

Показывает Text Draw с данным ID всем игрокам.

Синтаксис

TextDrawShowForAll(Text:text);

Параметры функции

Text:text – ID Text Draw

Возможное использование:

```
1  new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
2  TextDrawShowForAll(Welcome);
```

TextDrawHideForAll

Скрывает Text Draw с данным ID от всех игроков.

Синтаксис

TextDrawHideForAll(Text:text);

Параметры функции

Text:text – ID Text Draw

Возможное использование:

```
1  new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
2  TextDrawHideForAll(Welcome);
```

TextDrawSetString

Изменяет текст Text Draw.

Синтаксис

TextDrawSetString(Text:text, string[]);

Параметры функции

Text:text – ID Text Draw
String[] – Новый текст

Возможное использование:

```
1  new Text:Welcome = TextDrawCreate(320.0, 260.0, "Welcome to my server!");
2  TextDrawSetString(Welcome,"Goodbye");
```

Функции для работы с зонами банд

GangZoneCreate

Создает новую зону в памяти сервера. Функция возвращает ID созданной зоны.

Синтаксис

GangZoneCreate(Float:minx, Float:miny, Float:maxx, Float:maxy);

Параметры функции

Float:minx – Координата нижней границы зоны по оси x.
Float:miny – Координата нижней границы зоны по оси y.
Float:maxx – Координата верхней границы зоны по оси x.
Float:maxy – Координата верхней границы зоны по оси y.

Возможное использование:

```
1  new grovestreet = GangZoneCreate(1980.0, 280.0, 2110.0, 480.0);
```

GangZoneDestroy

Стирает созданную зону из памяти сервера.

Синтаксис

GangZoneDestroy(zone);

Параметры функции

zone – ID зоны которую нужно стереть

Возможное использование:

```
1  new grovestreet = GangZoneCreate(1980.0, 280.0, 2110.0, 480.0);
2  GangZoneDestroy(grovestreet);
```

GangZoneShowForPlayer

Показать зону для игрока.

Синтаксис

GangZoneShowForPlayer(playerid, zone, color);

Параметры функции

playerid – ID игрока, которому нужно показать зону

zone – ID зоны которую нужно показать
color – цвет зоны.

Возможное использование:

```
new grovestreet = GangZoneCreate(1980.0, 280.0, 2110.0, 480.0);
GangZoneShowForPlayer(playerid, grovestreet, 0xAAFFDDAA);
```

GangZoneHideForPlayer

Скрыть зону от игрока.

Синтаксис

GangZoneHideForPlayer(playerid, zone);

Параметры функции

playerid – ID игрока, которому нужно показать зону
zone – ID зоны которую нужно показать

Возможное использование:

```
new grovestreet = GangZoneCreate(1980.0, 280.0, 2110.0, 480.0);
GangZoneHideForPlayer(playerid, grovestreet);
```

GangZoneShowForAll

Показать зону для всех игроков.

Синтаксис

GangZoneShowForAll(zone, color);

Параметры функции

zone – ID зоны которую нужно показать
color – цвет зоны.

Возможное использование:

```
new grovestreet = GangZoneCreate(1980.0, 280.0, 2110.0, 480.0);
GangZoneShowForAll(grovestreet, 0xAAFFDDAA);
```

GangZoneHideForAll

Скрыть зону от всех игроков.

Синтаксис

GangZoneShowForAll(zone);

Параметры функции

zone – ID зоны которую нужно показать

Возможное использование:

```
new grovestreet = GangZoneCreate(1980.0, 280.0, 2110.0, 480.0);
GangZoneHideForAll(grovestreet);
```

GangZoneFlashForPlayer

Показать зону мигающей для игрока.

Синтаксис

GangZoneFlashForPlayer(playerid, zone, flashcolor);

Параметры функции

playerid – ID игрока
zone – ID зоны
flashcolor – цвет мигающей зоны.

Возможное использование:

```
1  new grovestreet = GangZoneCreate(1980.0, 280.0, 2110.0, 480.0);
2  GangZoneFlashForPlayer(playerid, grovestreet, 0xAAFFDDAA);
```

GangZoneFlashForAll

Показать зону мигающей для всех игроков.

Синтаксис

GangZoneFlashForAll(zone, flashcolor);

Параметры функции

zone – ID зоны
flashcolor – цвет мигающей зоны.

Возможное использование:

```
1  new grovestreet = GangZoneCreate(1980.0, 280.0, 2110.0, 480.0);
2  GangZoneFlashForAll(grovestreet, 0xAAFFDDAA);
```

GangZoneStopFlashForPlayer

Остановить мигание зоны для игрока.

Синтаксис

GangZoneStopFlashForPlayer(playerid, zone);

Параметры функции

playerid – ID игрока
zone – ID зоны

Возможное использование:

```
1  new grovestreet = GangZoneCreate(1980.0, 280.0, 2110.0, 480.0);
2  GangZoneStopFlashForPlayer(playerid, grovestreet);
```

GangZoneStopFlashForAll

Остановить мигание зоны для всех игроков.

Синтаксис

GangZoneStopFlashForAll(zone);

Параметры функции

zone – ID зоны

Возможное использование:

```
1  new grovestreet = GangZoneCreate(1980.0, 280.0, 2110.0, 480.0);
2  GangZoneStopFlashForAll(grovestreet);
```

Функции для работы с 3D текстом

Create3DTextLabel

Создает 3D надпись. Функция возвращает ID созданой надписи.

Синтаксис

Create3DTextLabel(text[], color, Float:X, Float:Y, Float:Z, Float:DrawDistance, virtualworld, testLOS=0);

Параметры функции

text[] – строка с текстом
color – цвет текста
Float:X – Координата X позиции 3D текста на карте.
Float:Y – Координата Y позиции 3D текста на карте.
Float:Z – Координата Z позиции 3D текста на карте.
DrawDistance - Максимальное расстояние видимости
VirtualWorld – ID виртуального мира
testLOS - Если 0 - то этот текст будет виден сквозь объекты, если 1 – нет

Возможное использование:

```
1  Create3DTextLabel( "3D текст", 0x008080FF,  1958.3783, 1343.1572, 15.3746, 20.0, 0, 0);
```

Delete3DTextLabel

Удаляет созданную 3D надпись.

Синтаксис

Delete3DTextLabel(Text3D:id);

Параметры функции

Text3D:id – ID 3D текста

Возможное использование:

```
1  new Text3D:text = Create3DTextLabel( "3D текст", 0x008080FF,  1958.3783, 1343.1572, 15.3746, 20.0, 0, 0);
2  Delete3DTextLabel(text);
```

Attach3DTextLabelToPlayer

Прикрепляет 3D текст к указанному игроку.

Синтаксис

Attach3DTextLabelToPlayer(Text3D:id, playerid, Float:OffsetX, Float:OffsetY, Float:OffsetZ);

Параметры функции

Text3D:id – ID 3D текста
playerid – ID игрока, которому нужно прикрепить текст.
OffsetX – Смещение по оси X, относительно центра игрока (сам игрок находится в 0.0).
OffsetY – Смещение по оси Y, относительно центра игрока (сам игрок находится в 0.0)
OffsetZ – Смещение по оси Z, относительно центра игрока (сам игрок находится в 0.0)

Возможное использование:

```
1  new Text3D:text = Create3DTextLabel( "3D текст", 0x008080FF,  1958.3783, 1343.1572, 15.3746, 20.0, 0, 0);
2  Attach3DTextLabelToPlayer(text, playerid, 0.0, 0.0, 0.5);
```

Attach3DTextLabelToVehicle

Прикрепляет 3D текст к указанному транспорту.

Синтаксис

Attach3DTextLabelToVehicle(Text3D:id, vehicleid, Float:OffsetX, Float:OffsetY, Float:OffsetZ);

Параметры функции

Text3D:id – ID 3D текста
vehicleid – ID транспорта.
OffsetX – Смещение по оси X, относительно центра транспорта (сам транспорт находится в 0.0).
OffsetY – Смещение по оси Y, относительно центра транспорта (сам транспорт находится в 0.0)
OffsetZ – Смещение по оси Z, относительно центра транспорта (сам транспорт находится в 0.0)

Возможное использование:

```
1  new Text3D:text = Create3DTextLabel( "3D текст", 0x008080FF,  1958.3783, 1343.1572, 15.3746, 20.0, 0, 0);
2  Attach3DTextLabelToVehicle(text, vehicleid, 0.0, 0.0, 0.5);
```

Update3DTextLabelText

Изменяет некоторые параметры 3Д текста, который ранее был создан для какого-то игрока.

Синтаксис

Update3DTextLabelText(Text3D:id, color, text[]);

Параметры функции

playerid – ID игрока, для которого он был создан.
PlayerText3D:id – ID 3D текста.
color - Новый цвет.
text[] - Новый текст.

Возможное использование:

```
1  new Text3D:text = Create3DTextLabel( "3D текст", 0x008080FF,  1958.3783, 1343.1572, 15.3746, 20.0, 0, 0);
2  Update3DTextLabelText(text, 0xFFFFFFFF, "New text.");
```

CreatePlayer3DTextLabel

Создает 3D текст, который будет виден только указанному игроку. Функция возвращает ID созданного текста.

Синтаксис

CreatePlayer3DTextLabel(playerid, text[], color, Float:X, Float:Y, Float:Z, Float:DrawDistance, attachedplayer=INVALID_PLAYER_ID, attachedvehicle=INVALID_VEHICLE_ID, testLOS=0);

Параметры функции

playerid – ID игрока, которой сможет видеть этот 3D текст.
text[] - Строка с текстом.
color - Цвет текста
Float:X - Координата X на карте ИЛИ X смещение от центра игрока, если 3D текст будет прикреплен
Float:Y - Координата Y на карте ИЛИ Y смещение от центра игрока, если 3D текст будет прикреплен
Float:Z - Координата Z на карте ИЛИ Z смещение от центра игрока, если 3D текст будет прикреплен
DrawDistance - Максимальное расстояние видимости
attachedplayer - ID игрока, к которому нужно прикрепить 3D текст ИЛИ INVALID_PLAYER_ID, если прикреплять не нужно)
attachedvehicle - ID транспорта, к которому нужно прикрепить 3D текст ИЛИ INVALID_VEHICLE_ID, если прикреплять не нужно)
testLOS - Если 0 - то этот текст будет виден сквозь объекты, если 1 - нет.

Возможное использование:

```
new Text3D:text = CreatePlayer3DTextLabel(playerid, "3D текст", 0x008080FF,  1958.3783, 1343.1572, 15.3746, 20.0, 0, 0);
```

DeletePlayer3DTextLabel

Удаляет 3D текст, был создан только для одного игрока. Функция возвращает ID созданного текста.

Синтаксис

DeletePlayer3DTextLabel(playerid, PlayerText3D:id);

Параметры функции

playerid – ID игрока, для которого этот 3D текст был создан.
PlayerText3D:id – ID самого 3D текста.

Возможное использование:

```
new Text3D:text = CreatePlayer3DTextLabel(playerid, "3D текст", 0x008080FF,  1958.3783, 1343.1572, 15.3746, 20.0, 0, 0);
DeletePlayer3DTextLabel(playerid, text);
```

UpdatePlayer3DTextLabel

Изменяет некоторые параметры 3D текста, который ранее был создан для какого-то игрока.

Синтаксис

DeletePlayer3DTextLabel(playerid, PlayerText3D:id);

Параметры функции

playerid – ID игрока, для которого он был создан.
PlayerText3D:id – ID 3D текста.
color - Новый цвет.
text[] - Новый текст.

Возможное использование:

```
new Text3D:text = CreatePlayer3DTextLabel(playerid, "3D текст", 0x008080FF,  1958.3783, 1343.1572, 15.3746, 20.0, 0, 0);
UpdatePlayer3DTextLabelText(playerid, text, 0xFFFFFFFF, "Hello World");
```

Функции A_SAMPDB.INC

Основные функции

db_open

Открывает базу данных для чтения. Функция возвращает ID базы данных, переменную типа BD.

Синтаксис

db_open(name[]);

Параметры функции

name[] – Имя файла в котором содержится база данных.

Возможное использование:

```
1  new DB:playersstats = db_open("stats.db");
```

db_close

Закрывает базу.

Синтаксис

db_close(DB:db);

Параметры функции

DB:db – Имя базы данных которую нужно закрыть.

Возможное использование:

```
1  db_close("stats.db");
```

db_query

Отправляет запрос в базу данных. Возвращает результата запроса, переменную типа DBResult

Синтаксис

db_query(DB:db,query[]);

Параметры функции

DB:db – Имя базы в которую нужно отправить запрос.
query[] – текст запроса

Возможное использование:

```
1  new DBResult:myresult = db_query(playersstats, "CREATE TABLE accounts (money INTEGER, bank INTEGER)");
```

db_free_result

Очищает результат запроса.

Синтаксис

db_free_result(DBResult:dbresult);

Параметры функции

playerid – ID игрока, которому отправляется сообщение в чат.
color – HEX-код цвета, которым будет выделено сообщение.
const message[] – текст сообщения, которое вы хотите отправить.

Возможное использование:

```
1 db_free_result(myresult);
```

db_num_rows

Узнает количество столбцов таблицы в результате запроса.

Синтаксис

db_num_rows(DBResult:dbresult);

Параметры функции

DBResult:dbresult – Результат запроса, который нужно обработать.

Возможное использование:

```
1 db_num_rows(myresult);
```

db_next_row

Переходит в следующий столбец в результате запроса.

Синтаксис

db_next_row(DBResult:dbresult);

Параметры функции

DBResult:dbresult – Запрос в результате которого нужно перейти в следущий столбец.

Возможное использование:

```
1 db_next_row(myresult);
```

db_num_fields

Узнает количество строк в таблице в результате запроса.

Синтаксис

db_num_fields(DBResult:dbresult);

Параметры функции

DBResult:dbresult – Запрос в результате которого нужно узнать количество строк в таблице.

Возможное использование:

```
1 db_num_fields(myresult);
```

db_field_name

Позволяет узнать название поля результата по его порядковому номеру.

Синтаксис

db_field_name(DBResult:dbresult, field, result[], maxlength);

Параметры функции

DBResult:dbresult - Идентификатор результата.
field - Порядковый номер поля результата.
result[] - Буфер для хранения имени поля.
maxlength - Число копируемых в буфер символов.

Возможное использование:

```
1  db_field_name(dbresult, i, name, sizeof(name));
```

db_get_field

Переходит в следующий столбец в результате запроса.

Синтаксис

native db_get_field(DBResult:dbresult, field, result[], maxlength);

Параметры функции

DBResult:dbresult – запрос в результате которого нужно перейти в следующий столбец.
field – Порядковый номер поля результата
result[] – Буфер для хранения имени поля
maxlength – Число копируемых в буфер символов

Возможное использование:

```
1  db_get_field(dbresult, 0, buffer, sizeof(buffer));
```

db_get_field_assoc

Позволяет узнать содержимое поля результата по его имени.

Синтаксис

db_get_field_assoc(DBResult:dbresult, const field[], result[], maxlength);

Параметры функции

DBResult:dbresult – Идентификатор результата
const field[] – Имя поля
result[] – Буфер для хранения имени поля
maxlength – Число копируемых в буфер символов

Возможное использование:

```
1
```

Функции A_PLAYERS.INC

Основные функции

SetPlayerPos

Изменяет текущую позицию игрока

Синтаксис

SetPlayerPos(playerid, Float:x, Float:y, Float:z);

Параметры функции

playerid - ID игрока.
Float:x - Координата - X, для новой позиции игрока.
Float:y – Координата - Y, для новой позиции игрока.
Float:z – Координата - Z, для новой позиции игрока.

Возможное использование:

```
1  SetPlayerPos(playerid, 34.236, 934.323, 12.345);
```

GetPlayerPos

Получает текущую позицию игрока

Синтаксис

GetPlayerPos(playerid, &Float:x, &Float:y, &Float:z);

Параметры функции

playerid - ID игрока.
Float:x – переменная, куда будет записана Координата – X текущей позиции игрока.
Float:y – переменная, куда будет записана Координата – Y текущей позиции игрока.
Float:z – переменная, куда будет записана Координата – Z текущей позиции игрока.

Возможное использование:

```
1  new Float:posx, Float:posy, Float:posz;
2  GetPlayerPos(playerid, posx, posy, posz);
```

SetPlayerFacingAngle

Устанавливает текущий угол поворота игрока.

Синтаксис

SetPlayerFacingAngle(playerid,Float:ang);

Параметры функции

playerid - ID игрока.
Float:ang - Новый угол поворота игрока.

Возможное использование:

```
1  new Float:posa;
2  SetPlayerFacingAngle(playerid, posa);
```

SetPlayerInterior

Изменяет текущий интерьер игрока.

Синтаксис

SetPlayerInterior(playerid,interiorid);

Параметры функции

playerid - ID игрока.
interiorid – ID интерьера игрока.

Возможное использование:

```
1  SetPlayerInterior(playerid, 0);
```

GetPlayerInterior

Получает текущий интерьер игрока.

Синтаксис

GetPlayerInterior(playerid,interiorid);

Параметры функции

playerid - ID игрока.
interiorid – ID интерьера игрока.

Возможное использование:

```
1  GetPlayerInterior(playerid, 0);
```

SetPlayerHealth

Устанавливает текущий уровень здоровья игрока.

Синтаксис

SetPlayerHealth(playerid,Float:health);

Параметры функции

playerid - ID игрока.
Float:health - Новое здоровье, вещественное число или проинициализированная переменная с значением здоровья.

Возможное использование:

```
1  SetPlayerHealth(playerid, 95,5);
```

GetPlayerHealth

Получает текущий уровень здоровья игрока.

Синтаксис

GetPlayerHealth(playerid,Float:health);

Параметры функции

playerid - ID игрока.
Float:health – Переменная, куда записывается текущий уровень здоровья игрока.

Возможное использование:

```
1  new Float:hp;
2  GetPlayerHealth(playerid, hp);
```

SetPlayerArmour

Устанавливает текущий уровень брони игрока.

Синтаксис

SetPlayerArmour(playerid,Float:ang);

Параметры функции

playerid - ID игрока.

Float:ang – Значение уровня брони или инициализированная переменная со значением брони.

Возможное использование:

```
1  SetPlayerArmour(playerid, 95.5);
```

GetPlayerArmour

Получает текущий уровень брони игрока.

Синтаксис

GetPlayerArmour(playerid,Float:ang);

Параметры функции

playerid - ID игрока.
Float:ang – Значение уровня брони или инициализированная переменная со значением брони.

Возможное использование:

```
1  GetPlayerArmour(playerid, 95.5);
```

SetPlayerAmmo

Устанавливает количество патронов у оружия в определенном слоте.

Синтаксис

SetPlayerAmmo(playerid, weaponslot, ammo);

Параметры функции

playerid - ID игрока.
weaponslot – Слот оружия
ammo – Количество патронов

Возможное использование:

```
1  SetPlayerAmmo(playerid,8,150);
```

GetPlayerAmmo

Функция возвращает текущее количество патронов у оружия которое держит игрок.

Синтаксис

GetPlayerAmmo(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  new ammo = GetPlayerAmmo(playerid);
```

GetPlayerWeaponState

Функция возвращает состояние оружия игрока

Синтаксис

GetPlayerWeaponState(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  GetPlayerWeaponState(playerid);
```

GetPlayerTargetPlayer

Функция проверяет прицеливание игрока на другого игрока.
Эта функция введена в **SAMP 0.3d** и в ранних версиях работать не будет.

Синтаксис

GetPlayerTargetPlayer(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  GetPlayerTargetPlayer(playerid);
```

SetPlayerTeam

Используйте эту функцию для изменения команды игрока.
Игроки одной команды не смогут навредить друг другу.

Синтаксис

SetPlayerTeam(playerid, teamid);

Параметры функции

playerid - ID игрока.
teamid - ID команды, членом которой станет игрок.

Возможное использование:

```
1  SetPlayerTeam(playerid, TEAM_BALLAS);
```

GetPlayerTeam

Получает идентификатор команды игрока в которой он находится.

Синтаксис

SetPlayerTeam(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  GetPlayerTeam(playerid);
```

SetPlayerScore

Обновляет счет игрока до другого значения.

Синтаксис

SetPlayerScore(playerid,score);

Параметры функции

playerid - ID игрока.
score - Новый счет игрока.

Возможное использование:

```
1  SetPlayerScore(playerid, 10000);
```

GetPlayerScore

Функция возвращает текущий счет игрока.

Синтаксис

GetPlayerScore(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  new score = GetPlayerScore(playerid);
```

SetPlayerDrunkLevel

Устанавливает уровень опьянения игрока.
Функция работает только в SAMP 0.3

Синтаксис

SetPlayerDrunkLevel(playerid, level);

Параметры функции

playerid - ID игрока.
level – уровень опьянения игрока (после уровня 2000 – игрок бухой, максимальный – 50000).

Возможное использование:

```
1  SetPlayerDrunkLevel(playerid, 2000);
```

GetPlayerDrunkLevel

Получает текущий уровень опьянения игрока.
Функция работает только в SAMP 0.3 и выше.

Синтаксис

GetPlayerDrunkLevel(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  new drunklevel =  GetPlayerDrunkLevel(playerid);
```

SetPlayerColor

Устанавливает цвет маркера над головой игрока и на радаре.

Синтаксис

SetPlayerColor(playerid,color);

Параметры функции

playerid - ID игрока.
color – Код цвета или константа его заменяющая.

Возможное использование:

```
1  SetPlayerColor(playerid, 0xFFFF00AA);
```

GetPlayerColor

Возвращает текущий цвет маркера над головой игрока и на радаре.

Синтаксис

GetPlayerColor(playerid);

Параметры функции
playerid - ID игрока.
Возможное использование:

```
1  new color[10] = GetPlayerColor(playerid);
```

SetPlayerSkin

Устанавливает скин персонажа игроку.

Синтаксис
SetPlayerSkin(playerid, skinid);
Параметры функции
playerid - ID игрока. skinid - ID нового скина для игрока.
Возможное использование:

```
1  SetPlayerSkin(playerid, 121);
```

GetPlayerSkin

Функция возвращает текущий скин персонажа игрока.

Синтаксис
GetPlayerSkin(playerid);
Параметры функции
playerid - ID игрока.
Возможное использование:

```
1  new pSkin = GetPlayerSkin(playerid);
```

GivePlayerWeapon

Дает оружие игрока с определенным количеством патронов.

Синтаксис
GivePlayerWeapon(playerid, weaponid, ammo);
Параметры функции
playerid – ID игрока, которому вы хотите дать оружие. weaponid – ID оружия, даваемого игроку. ammo – Количество патронов, даваемое игроку вместе с оружием.
Возможное использование:

```
1  GivePlayerWeapon(playerid,26, 250);
```

ResetPlayerWeapons

Сбрасывает все оружие у игрока.

Синтаксис
ResetPlayerWeapons(playerid);
Параметры функции
playerid – ID игрока, у которого вы хотите убрать все оружие.

Возможное использование:

```
1  ResetPlayerWeapons(playerid);
```

SetPlayerArmedWeapon

Устанавливает оружие, которое должен держать игрок в текущий момент.

Синтаксис

SetPlayerArmedWeapon(playerid, weaponid);

Параметры функции

playerid – ID игрока, у которого вы хотите убрать все оружие.
weaponid – ID оружия которое нужно установить.

Возможное использование:

```
1  SetPlayerArmedWeapon(playerid, 0);
```

GetPlayerWeaponData

Получает ID оружия и количество патров в конкретном слоте у игрока.

Синтаксис

GetPlayerWeaponData(playerid, slot, &weapons, &ammo);

Параметры функции

playerid - ID игрока.
slot - Слот оружия, информацию о котором нужно получить (0-12).
&weapons – Переменная, в которую нужно записать ID оружия в данном слоте.
&ammo - Переменная, в которую нужно записать количество патронов в данном слоте.

Возможное использование:

```
1  new weapon, wAmmo;
2  GetPlayerWeaponData(playerid, 7, weapon, wAmmo);
```

GetPlayerWeaponData

Получает ID оружия и количество патров в конкретном слоте у игрока.

Синтаксис

GetPlayerWeaponData(playerid, slot, &weapons, &ammo);

Параметры функции

playerid - ID игрока.
slot - Слот оружия, информацию о котором нужно получить (0-12).
&weapons – Переменная, в которую нужно записать ID оружия в данном слоте.
&ammo - Переменная, в которую нужно записать количество патронов в данном слоте.

Возможное использование:

```
1  new weapon, wAmmo;
2  GetPlayerWeaponData(playerid, 7, weapon, wAmmo);
```

GivePlayerMoney

Эта функция дает игроку указанное количество денег.
Если дать игроку отрицательное количество денег, то количество денег
игрока уменьшится или если игрок без денег, тогда у него будет долг.

Синтаксис

GivePlayerMoney(playerid,money);

Параметры функции

playerid - ID игрока.
Money – Количество денег, которые вы хотите дать/отнять.

Возможное использование:

```
1  GivePlayerMoney(playerid,10000);
```

ResetPlayerMoney

Сбрасывает (обнуляет) количество денег игрока.

Синтаксис

ResetPlayerMoney(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  ResetPlayerMoney(playerid);
```

SetPlayerName

Устанавливает имя игрока.

Синтаксис

SetPlayerName(playerid, const name[]);

Параметры функции

playerid - ID игрока.
const name[] - Новое имя игрока или строка с новым именем.

Возможное использование:

```
1  SetPlayerName(playerid, " Player");
```

GetPlayerName

Получает имя игрока.

Синтаксис

GetPlayerName(playerid, const name[], len);

Параметры функции

playerid - ID игрока.
const name[] – Строка, в которую необходимо записать имя игрока
len – длина этой строки.

Возможное использование:

```
1  new playername[MAX_PLAYER_NAME]
2  GetPlayerName(playerid, playername, sizeof(playername));
```

GetPlayerMoney

Получает текущее количество денег у игрока.

Синтаксис

GetPlayerMoney(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  new money = GetPlayerMoney(playerid);
```

GetPlayerState

Получает текущее состояние игрока.

Синтаксис

GetPlayerState(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  new state = GetPlayerState(playerid);
```

GetPlayerIp

Получает IP адрес игрока.

Синтаксис

GetPlayerIp(playerid, name[], len);

Параметры функции

playerid - ID игрока.
name[] – Строка, в которую необходимо записать IP игрока
len – длина этой строки.

Возможное использование:

```
1  new playerip[20] = GetPlayerIp(playerid, playerip, sizeof(playerip));
```

GetPlayerPing

Получает пинг игрока.

Синтаксис

GetPlayerPing(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  new playerping = GetPlayerPing(playerid);
```

GetPlayerWeapon

Получает ID оружия, которое держит игрок в данный момент.

Синтаксис

GetPlayerWeapon(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  new pWeapon = GetPlayerWeapon(playerid);
```

GetPlayerKeys

Возвращает ID клавиши которую игрок нажимает в данный момент.

Синтаксис

GetPlayerKeys(playerid, &keys, &updown, &leftright);

Параметры функции

playerid - ID игрока.
keys - Переменная хранящая состояние нажатой клавиши
updown- Вверх или вниз аналоговое значение
leftright- Левая или правая аналоговое значение

Возможное использование:

```
1  new Keys,ud,lr;
2  GetPlayerKeys(playerid,Keys,ud,lr);
```

SetPlayerTime

Устанавливает игровое время.

Синтаксис

SetPlayerTime(playerid, hour, minute);

Параметры функции

playerid - ID игрока.
hour – Строка, в которую необходимо записать имя игрока
minute – длина этой строки.

Возможное использование:

```
1  SetPlayerTime(playerid,12,00);
```

GetPlayerTime

Функция возвращает текущее время на сервере.

Синтаксис

GetPlayerTime(playerid, &hour, &minute);

Параметры функции

playerid - ID игрока.
hour – Строка, в которую необходимо записать имя игрока
minute – длина этой строки.

Возможное использование:

```
1  new hour,minute;
2  GetPlayerTime(playerid, hour, minute);
```

TogglePlayerClock

Функция включает или выключает игровые часы.

Синтаксис

TogglePlayerClock(playerid, toggle);

Параметры функции

playerid - ID игрока.
toggle – Включить – 1 или выключить – 0 игровые часы.

Возможное использование:

```
1  TogglePlayerClock(playerid);
```

SetPlayerWeather

Функция устанавливает погоду на сервере.

Синтаксис

SetPlayerWeather(playerid, weather);

Параметры функции

playerid - ID игрока.
weather – ID погоды.

Возможное использование:

```
1  SetPlayerWeather(playerid, 12);
```

ForceClassSelection

Возвращает игрока в выбору класса.

Синтаксис

ForceClassSelection(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  ForceClassSelection(playerid);
```

SetPlayerWantedLevel

Устанавливает уровень розыска игрока.

Синтаксис

SetPlayerWantedLevel(playerid, level);

Параметры функции

playerid - ID игрока.
level - Уровень розыска (кол-во звезд).

Возможное использование:

```
1  SetPlayerWantedLevel(playerid, 6);
```

GetPlayerWantedLevel

Получает уровень розыска игрока.

Синтаксис

GetPlayerWantedLevel(playerid, level);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  new wLevel = GetPlayerWantedLevel(playerid);
```

SetPlayerFightingStyle

Устанавливает стиль боя игрока.

Синтаксис

SetPlayerFightingStyle(playerid, style);

Параметры функции

playerid - ID игрока.
style - возвращает

Возможное использование:

```
1  SetPlayerFightingStyle(playerid,5);
```

Стили боя:
4 - FIGHT_STYLE_NORMAL
5 - FIGHT_STYLE_BOXING
6 - FIGHT_STYLE_KUNGFU
7 - FIGHT_STYLE_KNEEHEAD
15 - FIGHT_STYLE_GRABKICK
26 - FIGHT_STYLE_ELBOW

GetPlayerWantedLevel

Функция возвращает ID стиля боя игрока.

Синтаксис

GetPlayerWantedLevel(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  new style = GetPlayerFightingStyle(playerid);
```

SetPlayerVelocity

Задаёт скорость движения игрока в определённом направлении.

Синтаксис

SetPlayerVelocity(playerid, Float:X, Float:Y, Float:Z);

Параметры функции

playerid – ID игрока.
Float:X – скорость движения игрока по направлению по X-координате.
Float:Y – скорость движения игрока по направлению по Y-координате.
Float:Z – скорость движения игрока по направлению по Z-координате.

Возможное использование:

```
1  SetPlayerVelocity(playerid,0.0,0.0,0.2);
```

GetPlayerVelocity

Получает скорость движения игрока в определённом направлении.

Синтаксис

GetPlayerVelocity(playerid, &Float:X, &Float:Y, &Float:Z);

Параметры функции

playerid – ID игрока.
Float:X – скорость движения игрока по направлению по X-координате.
Float:Y – скорость движения игрока по направлению по Y-координате.

Float:Z – скорость движения игрока по направлению по Z-координате.

Возможное использование:

```
new Float:velx, Float:vely, Float:velz;
GetPlayerVelocity(playerid, velx, vely, velz);
```

PlayAudioStreamForPlayer

Эта функция позволяет создать аудиопоток для игрока.
Все параметры, кроме "playerid" и "url", являются не обязательными!
Функция работает только в SAMP 0.3d и выше.

Синтаксис

PlayAudioStreamForPlayer(playerid, url[], Float:posX = 0.0, Float:posY = 0.0, Float:posZ = 0.0, Float:distance = 50.0, usepos = 0);

Параметры функции

playerid – ID игрока.
url[]– URL потока. Допустимые расширения mp3/ogg!
Float:PosX – X - координата где будет включен поток. По умолчанию 0.0. Не имеет эффекта при значении 1.
Float:PosY – Y - координата где будет включен поток. По умолчанию 0.0. Не имеет эффекта при значении 1.
Float:PosZ – Z - координата где будет включен поток. По умолчанию 0.0. Не имеет эффекта при значении 1.
Float:distance – Дистанция воспроизведения, точнее радиус.
usepos – Используйте позицию и дистанцию. По умолчанию (0).

Возможное использование:

```
PlayAudioStreamForPlayer(playerid, "http://muzyaka.on.ufanet.ru/mp3/russian_way.mp3",0.0, 0.0, 0.0,50.0,true);
```

StopAudioStreamForPlayer

Останавливает аудиопоток для игрока.
Функция работает только в SAMP 0.3d и выше.

Синтаксис

StopAudioStreamForPlayer(playerid);

Параметры функции

playerid – ID игрока.

Возможное использование:

```
StopAudioStreamForPlayer(playerid);
```

SetPlayerSkillLevel

Устанавливает навык владения оружием.

Синтаксис

SetPlayerSkillLevel(playerid, skill, level);

Параметры функции

playerid – ID игрока.
skill – ID оружия навык владения которым нужно установить
level – навык владения оружием (0-1000).

Возможное использование:

```
SetPlayerSkillLevel(playerid, WEAPONSKILL_SAWNOFF_SHOTGUN, 200);
```

GetPlayerSurfingVehicleID

Функция возвращает ID транспорта на котором стоит игрок.

Синтаксис

GetPlayerSurfingVehicleID(playerid);

Параметры функции

playerid – ID игрока.

Возможное использование:

```
1  GetPlayerSurfingVehicleID(playerid);
```

RemoveBuildingForPlayer

Функция удаляет стандартный объект.
Функция работает только в SAMP 0.3d и выше.

Синтаксис

RemoveBuildingForPlayer(playerid, modelid, Float:fX, Float:fY, Float:fZ, Float:fRadius);

Параметры функции

playerid – ID игрока.
modelid – ID модели, которую мы удаляем.
Float:fX – координата X на которой примерно находится обьект.
Float:fY – координата Y на которой примерно находится обьект.
Float:fZ – координата Z на которой примерно находится обьект.
Float:fRadius – Радиус, в котором обьект не будет виден.

Возможное использование:

```
1  RemoveBuildingForPlayer(playerid, 615, 0.0, 0.0, 0.0, 6000.0);
```

PutPlayerInVehicle

Функция сажает игрока в транспорт.

Синтаксис

PutPlayerInVehicle(playerid, vehicleid, seatid);

Параметры функции

playerid – ID игрока, которого посадить в транспорт.
vehicleid - ID транспорта, куда нужно посадить игрока
seatid - ID места для посадки

Возможное использование:

```
1  PutPlayerInVehicle(playerid, 0, 1);
```

GetPlayerVehicleID

Функция возвращает ID транспорта в котором сидит игрок.

Синтаксис

GetPlayerVehicleID(playerid);

Параметры функции

playerid – ID игрока, которого посадить в транспорт.

Возможное использование:

```
1  new veh = GetPlayerVehicleID(playerid);
```

GetPlayerVehicleSeat

Функция возвращает ID места в транспорте на котором сидит игрок.

Синтаксис

GetPlayerVehicleSeat(playerid);

Параметры функции

playerid – ID игрока, которого посадить в транспорт.

Возможное использование:

```
1  new seat = GetPlayerVehicleSeat(playerid);
```

RemovePlayerFromVehicle

Функция выкидывает игрока из машины.

Синтаксис

RemovePlayerFromVehicle(playerid);

Параметры функции

playerid – ID игрока, которого посадить в транспорт.

Возможное использование:

```
1  RemovePlayerFromVehicle(playerid);
```

TogglePlayerControllable

Функция включает или отключает управление игроком.

Синтаксис

TogglePlayerControllable(playerid, toggle);

Параметры функции

playerid – ID игрока, которого посадить в транспорт.
toggle – разрешить – 1, запретить – 0 двигаться игроку

Возможное использование:

```
1  TogglePlayerControllable (playerid, 0);
```

PlayerPlaySound

Функция проигрывает звук для игрока.

Синтаксис

PlayerPlaySound(playerid, soundid, Float:x, Float:y, Float:z);

Параметры функции

playerid – ID игрока
soundid – ID звука
Float:X – Координата X места где нужно воспроизвести звук
Float:Y – Координата Y места где нужно воспроизвести звук
Float:Z – Координата Z места где нужно воспроизвести звук

Возможное использование:

```
1  PlayerPlaySound(playerid,1025,0.0,0.0,0.0);
```

ApplyAnimation

Функция проигрывает анимацию игрока.

Синтаксис

ApplyAnimation(playerid, animlib[], animname[], Float:fDelta, loop, lockx, locky, freeze, time, forcesync = 0);

Параметры функции

playerid – ID игрока
animlib[] - Имя библиотеки анимаций.
animname[] - Имя анимации.
fS - Скорость анимации.
opt1 - Повторяется ли она циклически или нет 0/1.
opt2 - Параметр можно назвать Фиксация координаты X. Если 1, то после 1 цикла анимации координата X позиции игрока - будет выставлена как до анимации, если 0 - то координата X не будет меняться на первоначальную.
opt3 - Параметр можно назвать Фиксация координаты Y. Если 1, то после 1 цикла анимации координата Y позиции игрока - будет выставлена как до анимации, если 0 - то координата Y не будет меняться на первоначальную.
opt4 - Заморозить ли игрока после окончания анимации.
opt5 - Таймер выполнения анимации. Для бесконечной анимации используйте 0.

Возможное использование:

```
1  ApplyAnimation( playerid, "PED", "WALK_DRUNK", 4.1, 1, 1, 1, 1);
```

ClearAnimations

Функция останавливет анимацию игрока.

Синтаксис

ClearAnimations(playerid, forcesync = 0);

Параметры функции

playerid – ID игрока

Возможное использование:

```
1  ClearAnimations(playerid);
```

GetAnimationName

Функция останавливет анимацию игрока.

Синтаксис

GetAnimationName(index, animlib[], len1, animname[], len2);

Параметры функции

index - индекс анимации(сам без понятия что это :D)
animlib[] - переменная, в которую мы запишем "библиотеку" анимации
len1 - размер переменной animlib[]
animname[] - переменная, в которую мы запишем название анимации
len2 - размер переменной animname[]

Возможное использование:

```
1  new animlib[32];
2  new animname[32];
3  GetAnimationName(GetPlayerAnimationIndex(playerid),animlib,32,animname,32);
```

GetPlayerSpecialAction

Функция возвращает ID специального действия игрока.

Синтаксис

GetPlayerSpecialAction(playerid);

Параметры функции

playerid – ID игрока

Возможное использование:

```
1  new action = GetPlayerSpecialAction(playerid);
```

SetPlayerSpecialAction

Проигрывает специальное действие для игрока.

Синтаксис

SetPlayerSpecialAction(playerid,actionid);

Параметры функции

playerid – ID игрока
actionid – ID специального действия

Возможное использование:

```
1  SetPlayerSpecialAction(playerid, SPECIAL_ACTION_USECELLPHONE);
```

SetPlayerCheckpoint

Устанавливает чекпоинт для игрока.

Синтаксис

SetPlayerCheckpoint(playerid, Float:x, Float:y, Float:z, Float:size);

Параметры функции

playerid - ID игрока.
Float:x – Новая координата - X, для чекпоинта.
Float:y – Новая координата - Y, для чекпоинта.
Float:z – Новая координата - Z, для чекпоинта.
Float:size – Размер чекпоинта.

Возможное использование:

```
1  SetPlayerCheckpoint(playerid, 354.12, 932.12, 19.34, 2.0);
```

DisablePlayerCheckpoint

Удаляет чекпоинт для игрока.

Синтаксис

DisablePlayerCheckpoint(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  DisablePlayerCheckpoint(playerid);
```

SetPlayerRaceCheckpoint

Устанавливает гоночный чекпоинт для игрока.

Синтаксис

SetPlayerRaceCheckpoint(playerid, type, Float:x, Float:y, Float:z, Float:nextx, Float:nexty, Float:nextz, Float:size);

Параметры функции

playerid - ID игрока.
type – Тип гоночного чекпойнта.
Float:x – Новая координата - X, для чекпоинта.
Float:y – Новая координата - Y, для чекпоинта.
Float:z – Новая координата - Z, для чекпоинта.
Float:nextx - Координата - X, куда будет смотреть стрелка чекпоинта.
Float: nexty – Координата - Y, куда будет смотреть стрелка чекпоинта.
Float: nextz – Координата - Z, куда будет смотреть стрелка чекпоинта.
Float:size – Размер чекпоинта.

Возможное использование:

```
1  SetPlayerRaceCheckpoint(playerid, 3,  354.12, 932.12, 19.34, 368.4, 898.2, 19.5, 2.0);
```

DisablePlayerRaceCheckpoint

Удаляет гоночный чекпоинт для игрока.

Синтаксис

DisablePlayerRaceCheckpoint(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  DisablePlayerRaceCheckpoint(playerid);
```

SetPlayerWorldBounds

Эта функция может быть использована для изменения границ
для игрока в игровом мире, мест, до куда он может добраться.

Синтаксис

SetPlayerWorldBounds(playerid,Float:x_max,Float:x_min,Float:y_max,Float:y_min);

Параметры функции

playerid - ID игрока.
Float:x_max - Макс. x-координата того, куда игрок может попасть.
Float:x_min - Мин. x-координата того, куда игрок может попасть.
Float:y_max - Макс. y-координата того, куда игрок может попасть.
Float:y_min - Мин. y-координата того, куда игрок может попасть.

Возможное использование:

```
1  SetPlayerWorldBounds(playerid, 20.0, 0.0, 20.0, 0.0);
```

SetPlayerMarkerForPlayer

Изменяет цвет маркера определенного игрока для игрока.

Синтаксис

SetPlayerMarkerForPlayer(playerid, showplayerid, color);

Параметры функции

playerid - ID игрока, который будет видеть новый цвет маркера у другого игрока.
showplayerid - ID игрока, цвет которого будет видеть другому игроку.
color – цвет маркера.

Возможное использование:

```
1  SetPlayerMarkerForPlayer( 42,1,0xFF0000FF );
```

ShowPlayerNameTagForPlayer

Скрывает или показывает маркер и тег имени определенного игрока для игрока.

Синтаксис

ShowPlayerNameTagForPlayer(playerid, showplayerid, show);

Параметры функции

playerid - ID игрока, который будет видеть или не видеть маркер и тег имени другого игрока.
showplayerid - ID игрока, маркер и тег имени которого нужно скрыть.
show – показать 1 или не показываеть 0 маркер или тег имени другого игрока.

Возможное использование:

```
1  ShowPlayerNameTagForPlayer(playerid, showplayerid, 0);
```

SetPlayerMapIcon

Устанавливает для игрока иконку на радаре и карте.

Синтаксис

SetPlayerMapIcon(playerid, iconid, Float:x, Float:y, Float:z, markertype, color, style = MAPICON_LOCAL);

Параметры функции

playerid - ID игрока.
iconid - ID для данной иконки.
Float:x - Координата - X для положения иконки.
Float:y - Координата - Y для положения иконки.
Float:z - Координата - Z для положения иконки.
markertype - Тип (модель) иконки.
color - Цвет иконки.

Возможное использование:

```
1  new ico = SetPlayerMapIcon(playerid, 1, 2033.6526,-1405.4602,17.2334, 22, 0xFFFFFFAA);
```

RemovePlayerMapIcon

Удаляет для игрока иконку на радаре и карте.

Синтаксис

RemovePlayerMapIcon(playerid, iconid);

Параметры функции

playerid - ID игрока.
iconid - ID для данной иконки.

Возможное использование:

```
1  new ico = SetPlayerMapIcon(playerid, 1, 2033.6526,-1405.4602,17.2334, 22, 0xFFFFFFAA);
2  RemovePlayerMapIcon(playerid, ico);
```

SetPlayerCameraPos

Устанавливает позицию камеры игрока.

Синтаксис

SetPlayerCameraPos(playerid,Float:x, Float:y, Float:z);

Параметры функции

playerid - ID игрока.
Float:x - Координата - X, куда будет смотреть камера.
Float:y – Координата - Y, куда будет смотреть камера.
Float:z – Координата - Z, куда будет смотреть камера.

Возможное использование:

```
1 SetPlayerCameraPos(playerid, 652.23, 457.21, 10.84);
```

SetPlayerCameraLookAt

Удаляет позицию куда смотрит камера.

Синтаксис

SetPlayerCameraLookAt(playerid, Float:x, Float:y, Float:z);

Параметры функции

playerid - ID игрока.
Float:x – Координата - X, куда будет смотреть камера.
Float:y – Координата - Y, куда будет смотреть камера.
Float:z – Координата - Z, куда будет смотреть камера.

Возможное использование:

```
1 SetPlayerCameraLookAt(playerid, 652.23, 457.21, 10.84);
```

SetCameraBehindPlayer

Возвращает камеру в положение за игроком.

Синтаксис

SetCameraBehindPlayer(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1 SetCameraBehindPlayer(playerid);
```

GetPlayerCameraPos

Получает текущее положение камеры игрока.

Синтаксис

GetPlayerCameraPos(playerid, &Float:x, &Float:y, &Float:z);

Параметры функции

playerid - ID игрока.
&Float:x – Переменная куда будет записана координата - X.
&Float:y – Переменная куда будет записана координата - Y.
&Float:z – Переменная куда будет записана координата - Z.

Возможное использование:

```
1 SetCameraBehindPlayer(playerid);
```

GetPlayerCameraMode

Функция возвращающая текущий режим камеры игрока.
Функция работает только в **SAMP 0.3c** и выше

Синтаксис

GetPlayerCameraMode(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  GetPlayerCameraMode(playerid);
```

GetPlayerCameraFrontVector

Функция возвращает направлении камеры направление камеры игрока относительно координат полученных при помощи GetPlayerCameraPos

Синтаксис

GetPlayerCameraFrontVector(playerid, &Float:x, &Float:y, &Float:z);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  new Float:fVX, Float:fVY, Float:fVZ;
2  GetPlayerCameraFrontVector(playerid, fVX, fVY, fVZ);
```

GetPlayerCameraFrontVector

Функция проверяет игрока на подключение к серверу.

Синтаксис

IsPlayerConnected(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  If(IsPlayerConnected(playerid))
```

IsPlayerInVehicle

Функция проверяет сидит ли игрок в указанном транспорте с определенным ID.

Синтаксис

IsPlayerInVehicle(playerid, vehicleid);

Параметры функции

playerid - ID игрока.
vehicleid - ID транспорта.

Возможное использование:

```
1  If(IsPlayerInVehicle (playerid,1242))
```

IsPlayerInAnyVehicle

Функция проверяет сидит ли игрок в любом транспорте.

Синтаксис

IsPlayerInAnyVehicle(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  If(IsPlayerInAnyVehicle(playerid))
```

IsPlayerInCheckpoint

Функция проверяет встал ли игрок в указанный чекпоинт.

Синтаксис

IsPlayerInCheckpoint(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  If(IsPlayerInCheckpoint(playerid))
```

IsPlayerInRaceCheckpoint

Функция проверяет встал ли игрок в указанный гоночный чекпоинт.

Синтаксис

IsPlayerInRaceCheckpoint(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  If(IsPlayerInRaceCheckpoint(playerid))
```

SetPlayerVirtualWorld

Функция устанавливает ID виртуального мира игроку.

Синтаксис

SetPlayerVirtualWorld(playerid, worldid);

Параметры функции

playerid - ID игрока.
worldid – ID виртуального мира.

Возможное использование:

```
1  SetPlayerVirtualWorld(playerid, 1);
```

GetPlayerVirtualWorld

Функция возвращает ID виртуального мира игрока.

Синтаксис

GetPlayerVirtualWorld(playerid);

Параметры функции

playerid - ID игрока.

Возможное использование:

```
1  SetPlayerVirtualWorld(playerid, 1);
```

TogglePlayerSpectating

Функция включает или выключает режим наблюдения у игрока.

Синтаксис

TogglePlayerSpectating(playerid, toggle);

Параметры функции

playerid - ID игрока.
toggle – включить – 1 или выключить – 0 режим спектатора.

Возможное использование:

```
1 TogglePlayerSpectating(playerid, 1);
```

PlayerSpectatePlayer

Функция включает или выключает режим наблюдения за определенным игроком для игрока.

Синтаксис

PlayerSpectatePlayer(playerid, targetplayerid, mode = SPECTATE_MODE_NORMAL);

Параметры функции

playerid - ID игрока, который будет осуществлять наблюдение.
targetplayerid - ID игрока, за которым будет осуществляться наблюдение.
mode - Режим наблюдения.

Возможное использование:

```
1 PlayerSpectatePlayer(playerid, 4, SPECTATE_MODE_NORMAL);
```

PlayerSpectateVehicle

Функция включает или выключает режим наблюдения за определенным транспортом для игрока.

Синтаксис

PlayerSpectateVehicle(playerid, targetvehicleid, mode = SPECTATE_MODE_NORMAL);

Параметры функции

playerid - ID игрока, который будет осуществлять наблюдение.
targetvehicleid - ID игрока, за которым будет осуществляться наблюдение.
mode - Режим наблюдения.

Возможное использование:

```
1 PlayerSpectateVehicle(playerid, 4, SPECTATE_MODE_NORMAL);
```

Функции A_OBJECTS.INC

Основные функции

CreateObject

Создает объект. Функция возвращает ID созданного объекта.

Синтаксис

CreateObject(modelid, Float:X, Float:Y, Float:Z, Float:rX, Float:rY, Float:rZ, Float:DrawDistance = 0.0);

Параметры функции

modelid - ID модели объекта.
Float:X - Координата X объекта.
Float:Y - Координата Y объекта.
Float:Z - Координата Z объекта.
Float:rX - Угол поворота объекта вокруг оси X.
Float:rY - Угол поворота объекта вокруг оси Y.
Float:rZ - Угол поворота объекта вокруг оси Z.

Возможное использование:

```
1  new objectid = CreateObject(11452, 1154.0, 1024.8, 45.2, 0, 0, 28.4);
```

SetObjectPos

Изменяет положение объекта.

Синтаксис

SetObjectPos(objectid, Float:X, Float:Y, Float:Z);

Параметры функции

objectid - ID объекта.
Float:X - Координата X нового положения объекта.
Float:Y - Координата Y нового положения объекта.
Float:Z - Координата Z нового положения объекта.

Возможное использование:

```
1  SetObjectPos(objectid, 1154.0, 1024.8, 45.2);
```

GetObjectPos

Узнает положение объекта.

Синтаксис

GetObjectPos(objectid, &Float:X, &Float:Y, &Float:Z);

Параметры функции

objectid - ID объекта.
&Float:X - Переменная, в которую записывается координата X положения объекта.
&Float:Y - Переменная, в которую записывается координата Y положения объекта.
&Float:Z - Переменная, в которую записывается координата Z положения объекта.

Возможное использование:

```
1  new Float:X, Float:Y, Float:Z;
2  GetObjectPos(objectid, X, Y, Z);
```

SetObjectRot

Изменяет поворот объекта.

Синтаксис

SetObjectRot(objectid, Float:RotX, Float:RotY, Float:RotZ);

Параметры функции

objectid - ID объекта.
RotX - Угол поворота объекта вокруг оси X.
RotY - Угол поворота объекта вокруг оси Y.
RotZ - Угол поворота объекта вокруг оси Z.

Возможное использование:

```
1  SetObjectRot(objectid, 0, 0, 28.4);
```

GetObjectRot

Узнает поворот объекта.

Синтаксис

SetObjectRot(objectid, Float:RotX, Float:RotY, Float:RotZ);

Параметры функции

objectid - ID объекта.
RotX - Угол поворота объекта вокруг оси X.
RotY - Угол поворота объекта вокруг оси Y.
RotZ - Угол поворота объекта вокруг оси Z.

Возможное использование:

```
1  new Float:X, Float:Y, Float:Z;
2  GetObjectRot(objectid, X, Y, Z);
```

GetObjectRot

Узнает поворот объекта.

Синтаксис

GetObjectRot(objectid, &Float:RotX, &Float:RotY, &Float:RotZ);

Параметры функции

objectid - ID объекта.
&Float:X - Переменная, в которую записывается угол поворота объекта вокруг оси X.
&Float:Y - Переменная, в которую записывается угол поворота объекта вокруг оси Y.
&Float:Z - Переменная, в которую записывается угол поворота объекта вокруг оси Z.

Возможное использование:

```
1  new Float:X, Float:Y, Float:Z;
2  GetObjectRot(objectid, X, Y, Z);
```

IsValidObject

Проверяет, существует ли объект.

Синтаксис

IsValidObject(objectid);

Параметры функции

objectid - ID объекта.

Возможное использование:

```
1  if (IsValidObject(objectid)) {
```

DestroyObject

Убирает объект.

Синтаксис

DestroyObject(objectid);

Параметры функции

objectid - ID объекта.

Возможное использование:

```
1  DestroyObject(objectid);
```

MoveObject

Двигает объект.

Синтаксис

MoveObject(objectid, Float:X, Float:Y, Float:Z, Float:Speed);

Параметры функции

objectid - ID объекта.
Float:X - Координата X точки, в которую нужно двигать объект.
Float:Y - Координата Y точки, в которую нужно двигать объект.
Float:Z - Координата Z точки, в которую нужно двигать объект.
Float:Speed - Скорость движения объекта.

Возможное использование:

```
1  MoveObject(objectid, 1254.0, 1128.0, 67.0, 20.0);
```

StopObject

Останавливает движущийся объект.

Синтаксис

StopObject(objectid);

Параметры функции

objectid - ID объекта.

Возможное использование:

```
1  StopObject(objectid);
```

CreatePlayerObject

Создает объект, который будет виден только указанному игроку.
Объект можно уничтожить с помощью **DestroyPlayerObject**.
Функция возвращает ID созданного объекта.

Синтаксис

CreatePlayerObject(playerid, modelid, Float:X, Float:Y, Float:Z, Float:rX, Float:rY, Float:rZ, Float:DrawDistance = 0.0);

Параметры функции

playerid – ID игрока, который сможет видеть этот объект.

modelid – ID модели объекта.
Float:X - Координата X на карте
Float:Y - Координата Y на карте
Float:Z - Координата Z на карте
Float:rX - Угол поворота объекта по оси X.
Float:rY - Угол поворота объекта по оси Y
Float:rZ - Угол поворота объекта по оси Z

Возможное использование:

```
1  new object = CreatePlayerObject (playerid, 2587, 2001.195679, 1547.113892, 14.283400, 0, 0, 96 );
```

SetPlayerObjectPos

Устанавливает положение объекта игрока.

Синтаксис

SetPlayerObjectPos(playerid, objectid, Float:X, Float:Y, Float:Z);

Параметры функции

playerid – ID игрока, который сможет видеть этот объект.
modelid – ID модели объекта.
Float:X - Координата X на карте
Float:Y - Координата Y на карте
Float:Z - Координата Z на карте

Возможное использование:

```
1  SetPlayerObjectPos(playerid, objectid, 2001.195679, 1547.113892, 14.283400);
```

GetPlayerObjectPos

Получает координаты текущего положения объекта игрока и записывает их в переменные.

Синтаксис

GetPlayerObjectPos(playerid, objectid, &Float:X, &Float:Y, &Float:Z);

Параметры функции

playerid – ID игрока, который сможет видеть этот объект.
modelid – ID модели объекта.
&Float:X - Переменная, в которую записывается, координата X текущего положения объекта игрока.
&Float:Y - Переменная, в которую записывается, координата Y текущего положения объекта игрока.
&Float:Z - Переменная, в которую записывается, координата Z текущего положения объекта игрока.

Возможное использование:

```
1  new Float:x, Float:y, Float:z;
2  GetPlayerObjectPos(playerid, objectid, x, y, z);
```

Функции A_VEHICLES.INC

Основные функции

ChangeVehicleColor

Эта функция позволяет изменить первичный и вторичный цвет уже созданного транспорта.
Хотя для некоторых видов транспорта есть и третий цвет, но его нельзя изменить с помощью скриптинга.

Синтаксис

ChangeVehicleColor(vehicleid, color1, color2);

Параметры функции

vehicleid – ID уже созданного транспорта
color1 – ID нового первичного цвета
color2 – ID нового вторичного цвета

Возможное использование:

```
1  ChangeVehicleColor (vehicleid, 0,1);
```

ChangeVehiclePaintjob

Эта функция может устанавливать наклейки на транспорт основные
ID наклеек – это 1,2 и 3,однако, иногда используются также и 0,4 и 5.

Синтаксис

ChangeVehiclePaintjob(vehicleid, paintjobid);

Параметры функции

vehicleid – ID уже созданного транспорта, на котором будет помещена наклейка.
paintjobid – ID наклейки. 0 – удалит все наклейки.

Возможное использование:

```
1  new veh;
2  veh = GetPlayerVehicleID(playerid);
3  ChangeVehiclePaintjob(veh,2);
```

AddVehicleComponent

Добавляет новый компонент в транспорте, например, нитро.

Синтаксис

AddVehicleComponent(vehicleid, componentid);

Параметры функции

vehicleid - ID транспорта, к которому добавляется компонент.
componentid - ID добавляемого компонента.

Возможное использование:

```
1  AddVehicleComponent(420, 1095);
```

DestroyVehicle

Убирает транспорт с данным ID с карты (уничтожает).

Синтаксис

DestroyVehicle(vehicleid);

Параметры функции

vehicleid - ID транспорта

Возможное использование:

```
1  DestroyVehicle(vehicleid);
```

GetVehicleHealth

Получает текущее здоровье транспорта.

Синтаксис

GetVehicleHealth(vehicleid, &Float:health);

Параметры функции

vehicleid - ID транспорта
&Float:health - Переменная, в которую записывается здоровье транспортного средства.

Возможное использование:

```
1  new Float:vehhp;
2  GetVehicleHealth(vehicleid, vehhp);
```

GetVehicleModel

Узнает ID модели данного транспортного средства.

Синтаксис

GetVehicleModel(vehicleid);

Параметры функции

vehicleid - ID транспорта

Возможное использование:

```
1  new Float:vehhp;
2  GetVehicleHealth(vehicleid, vehhp);
```

GetVehicleModel

Узнает ID модели данного транспортного средства.

Синтаксис

GetVehicleModel(vehicleid);

Параметры функции

vehicleid - ID транспорта

Возможное использование:

```
1  new modelid = GetVehicleModel(vehicleid);
```

GetVehiclePos

Получает координаты текущего положения транспорта и сохраняет
их в переменных, указанных в трех последующих аргументах.

Синтаксис

GetVehiclePos(vehicleid, &Float:x, &Float:y, &Float:z);

Параметры функции

vehicleid - ID транспорта
&Float:x - Переменная, в которую записывается x-координата.
&Float:y - Переменная, в которую записывается y-координата.
&Float:z - Переменная, в которую записывается z-координата.

Возможное использование:

```
1  GetVehiclePos(vehicleid, 1958.3783, 1343.1572, 15.3746);
```

GetVehicleVirtualWorld

Узнает ID виртуального мира, в котором находится данное транспортное средство.

Синтаксис

GetVehicleVirtualWorld(vehicleid);

Параметры функции

vehicleid - ID транспорта

Возможное использование:

```
1  new vehworld = GetVehicleVirtualWorld(vehicleid);
```

GetVehicleZAngle

Получает угол поворота транспортного средства.

Синтаксис

GetVehicleZAngle(vehicleid, &Float:z_angle);

Параметры функции

vehicleid - ID транспорта
&Float:z_angle - Переменная, в которую записывается угол.

Возможное использование:

```
1  new Float:vAngle;
2  GetVehicleZAngle(vehicleid, vAngle);
```

LinkVehicleToInterior

Привязывает данное транспортное средство к определенному интерьеру.

Синтаксис

LinkVehicleToInterior(vehicleid, interiorid);

Параметры функции

vehicleid - ID транспорта
interiorid - ID интерьера, к которому нужно привязать транспорт.

Возможное использование:

```
1  LinkVehicleToInterior(vehicleid, 6);
```

SetVehicleHealth

Устанавливает здоровье транспортного средства. Полное здоровье - 1000.

Синтаксис

SetVehicleHealth(vehicleid, Float:health);

Параметры функции

vehicleid - ID транспорта
Float:health - Количество единиц здоровья, которое нужно установить.

Возможное использование:

```
1  SetPlayerHealth(vehicleid, 95.0);
```

SetVehicleNumberPlate

Устанавливает номерной знак для данного транспортного средства.

Синтаксис

SetVehicleNumberPlate(vehicleid, numberplate[]);

Параметры функции

vehicleid - ID транспорта
numberplate[] - Номерной знак.

Возможное использование:

```
1  SetVehicleNumberPlate(vehicleid, "N654RS");
```

SetVehicleParamsForPlayer

Изменяет указанные параметры транспорта для игрока.
Она позволяет установить стрелку над игроком или заблокировать двери.

Синтаксис

SetVehicleParamsForPlayer(vehicleid,playerid,objective,doorslocked);

Параметры функции

vehicleid - ID транспорта.
playerid - ID игрока.
objective - Должен ли быть транспорт целью ?
doorslocked - Хотите ли Вы, чтобы двери транспорта были заблокированы ?

Возможное использование:

```
1  SetVehicleParamsForPlayer(vehicleid, playerid, 1, 1);
```

SetVehiclePos

Изменяет позицию транспорта. Если кто-то находится в транспорте, то он телепортируется вместе с ним.
Важное замечание: Эта функция работает ТОЛЬКО в том случае, если кто-то побывал в транспорте с тех пор, как тот разместился в стартовой позиции.

Синтаксис

SetVehiclePos(vehicleid, Float:x, Float:y, Float:z);

Параметры функции

vehicleid - ID транспорта.
Float:x X-координата новой позиции транспорта.
Float:y Y-координата новой позиции транспорта.
Float:z Z-координата новой позиции транспорта.

Возможное использование:

```
1 SetVehiclePos(vehicleid, 343.46, 459.34, 19.23);
```

SetVehicleVirtualWorld

Помещает данное транспортное средство в виртуальный мир.

Синтаксис

SetVehicleVirtualWorld(vehicleid, worldid);

Параметры функции

vehicleid - ID транспорта.
worldid - ID виртуального мира.

Возможное использование:

```
1 SetVehicleVirtualWorld(vehicleid, 4);
```

SetVehicleToRespawn

Позволяет Вам переразметить в стартовую позицию указанный транспорт,
даже когда в это время им управляет какой-нибудь игрок.

Синтаксис

SetVehicleToRespawn(vehicleid);

Параметры функции

vehicleid - ID транспорта.

Возможное использование:

```
1 SetVehicletoRespawn(vehicleid);
```

SetVehicleZAngle

Эта функция позволяет Вам изменить угол поворота указанного транспорта.

Синтаксис

SetVehicleZAngle(vehicleid, Float:z_angle);

Параметры функции

vehicleid - ID транспорта.
Float:z_angle - Новый угол поворота транспорта.

Возможное использование:

```
1 SetVehicleZAngle(vehicleid, 270.0);
```

AttachTrailerToVehicle

Присоединяет прицеп к транспорту.

Синтаксис

AttachTrailerToVehicle(trailerid, vehicleid);

Параметры функции

trailerid - ID прицепа.
vehicleid - ID транспорта.

Возможное использование:

```
1 AttachTrailerToVehicle(5,1);
```

GetVehicleTrailer

Узнает ID прицепа прикрепленного к транспорту.

Синтаксис

GetVehicleTrailer(vehicleid);

Параметры функции

trailerid - ID прицепа.

Возможное использование:

```
1 GetVehicleTrailer(vehicleid);
```

Функции CORE.INC

Основные функции

random

Функция возвращает число между 0 и максимальным значением.

Синтаксис

random(max);

Параметры функции

max – Максимальное число для случайной последовательности.

Возможное использование:

```
1 random(12);
```

sizeof

Возвращает предопределенный размер переменной.

Синтаксис

sizeof(string[]);

Параметры функции

string[] – Переменная размер которой нужно возвратить.

Возможное использование:

```
1 sizeof(string);
```

Функции FILE.INC

Функции для работы с файлами

fblockread

Эта функция позволяет Вам прочитать данные из файла без кодирующих и завершающих символов (в двоичном формате)

Синтаксис

fblockread(handle, buffer[], size=sizeof buffer);

Параметры функции

handle - Используемый handle файла, открытый функцией fopen().
buffer[] - Буфер для сохранения прочитанных данных.
size=sizeof buffer - Число ячеек для чтения.

Возможное использование:

```
1  fblockread(gFile, string, 256);
```

fblocwrite

Записывает данные в файл в двоичном формате, игнорируя концы строк и кодировку.

Синтаксис

fblockwrite(handle, const buffer[], size=sizeof buffer);

Параметры функции

handle - Используемый handle файла, открытый функцией fopen().
const buffer[] – Данные для записи в файл.
size=sizeof buffer - Число ячеек для записи.

Возможное использование:

```
1  fblockwrite(gFile, "Save this data!", 256);
```

fclose

Закрывает handle файла, ранее открытый функцией fopen. Очень важно использовать эту функцию после того, как вы закончили чтение/запись!

Синтаксис

flose(handle);

Параметры функции

handle - Используемый handle файла, открытый функцией fopen().

Возможное использование:

```
1  fclose(gFile);
```

fexist

Проверяет, есть ли указанный файл в папке с Вашими скриптами.

Синтаксис

fexist(const pattern);

Параметры функции

const pattern - Имя файла для проверки на существование.

Возможное использование:

```
1 if(fexist("datafile.txt")){
```

fgetchar

Эта функция читает один символ из файла и сохраняет его в переменной, переданной по ссылке.

Синтаксис

fgetchar(handle, &value, utf=true);

Параметры функции

handle - используемый handle файла, открытый функцией fopen().
&value - переменная, в которую запишется символ.
utf8=true - прочитать символ в кодировке UTF8.

Возможное использование:

```
1 fgetchar(gFile, handle, false);
```

flength

Функция возвращает длину уже открытого файла.

Синтаксис

flength(handle);

Параметры функции

handle - используемый handle файла, открытый функцией fopen().

Возможное использование:

```
1 new filelength = flength(gFile);
```

fmatch

Эта функция проверяет, соответствует ли часть данного файла указанной строке.

Синтаксис

fmatch(name[], const pattern, index=0, size=sizeof name);

Параметры функции

name[] - имя проверяемого файла.
const pattern[] - шаблон соответствия.
index=0 - смещение для начала поиска.
size=sizeof name - количество символов, в которых будет производиться поиск.

Возможное использование:

```
1 fmatch("searchFile.txt", "Peter", 0);
```

fopen

Открывает указанный файл для чтения, записи или для обеих операций.
Эта функция нужна для большинства файловых функций. Важное замечание:
Эта функция может привести к вылету Вашей игры, когда папка с скриптами или файл
в ней не существуют. Функция возвращает - Handle файла.

Синтаксис

fopen(const pattern, mode=io_readwrite);

Параметры функции

const name[] - имя файла, который Вы хотите открыть.
mode=io_readwrite - режим, в котором вы хотите открыть файл.

Возможное использование:

```
1  new File:gFile = fopen("exampleFile.txt", io_readwrite);
```

fputchar

Эта функция записывает один символ в файл.

Синтаксис

fputchar(handle, &alue, utf=true);

Параметры функции

handle - используемый handle файла, открытый функцией fopen().
value - символ, который вы хотите записать.
utf8=true - прочитать символ в кодировке UTF8.

Возможное использование:

```
1  fputchar(gFile, 'e', false);
```

fread

Используйте эту функцию, если Вы хотите прочитать строку из файла.
Вам понадобится цикл for или while для чтения нескольких строк.

Синтаксис

fread(handle, &string, size=sizeof string, pack=false);

Параметры функции

handle - используемый handle файла, открытый функцией fopen().
&string[] - переданная по ссылке строка, в которую запишутся данные.
size=sizeof string - число байт для чтения.
pack=false - должна ли строка быть запакованной?

Возможное использование:

```
1  fread(gFile, string, sizeof(string));
```

fremove

Удаляет существующий файл из папки с Вашими скриптами.
Важное замечание: Эта функция может привести к вылету Вашей игры,
когда папка с скриптами или файл в ней не существуют.

Синтаксис

fremove(const name[]);

Параметры функции

const name[] – имя файла, который вы хотите удалить.

Возможное использование:

```
1  fremove("exampleFile.txt");
```

fseek

Изменяет текущую позицию в файле. Вы можете перемещаться по файлу вперед или назад.

Синтаксис

fseek(handle, position=0, whence=seek_start);

Параметры функции

handle - используемый handle файла, открытый функцией fopen().
position=0 - позиция для помещения файлового курсора.
whence=seek_start - способ, по которому нужно переместиться по файлу.

Возможное использование:

```
1  fseek(gFile, 25);
```

ftemp

Эта функция открывает файл в папке "tmp" или "temp" для чтения или записи.
Файл удалится после того, как Вы закроете его функцией fclose().
Важное замечание: Эта функция может привести к вылету Вашей игры,
когда папка с скриптами или файл в ней не существуют. Возвращает: Handle файла.

Синтаксис

fseek(handle, position=0, whence=seek_start);

Параметры функции

handle - используемый handle файла, открытый функцией fopen().
position=0 - позиция для помещения файлового курсора.
whence=seek_start - способ, по которому нужно переместиться по файлу.

Возможное использование:

```
1  new File:gFile = ftemp();
```

fwrite

Записывает указанную строку или линию в файл. Помните, что файл должен быть открыт для записи.

Синтаксис

fwrite(handle, const string[]);

Параметры функции

handle Используемый handle файла, открытый функцией fopen().
const string[] Строка, которую Вы хотите записать в файл.

Возможное использование:

```
1  fwrite(gFile, "This will be put in the file!");
```

Функции FLOAT.INC

Основные функции

float

Эта функция конвертирует целое число в вещественное.

Синтаксис

float (value);

Параметры функции

value – целое число, которое вы хотите конвертировать в вещественное.

Возможное использование:

```
1  new Float:fVar = float(122);
```

floatabs

Возвращает абсолютное значение вещественного числа.

Синтаксис

floatabs(value);

Параметры функции

value –Вещественное число.

Возможное использование:

```
1  new Float:fVar = floatabs(122.23);
```

floatadd

Вычисляет сумму двух вещественных чисел, то же самое, что и Float1 + Float2. Возвращает сумму чисел.

Синтаксис

floatadd(oper1, oper2);

Параметры функции

oper1 – Первое вещественное число.
oper2 – Второе вещественное число.

Возможное использование:

```
1  new Float:fSum = floatadd(122.45, 678.90);
```

floatcmp

Сравнивает два вещественных числа.

Синтаксис

floatcmp (oper1, oper2);

Параметры функции

oper1 – Первое вещественное число.
oper2 – Второе вещественное число.

Возможное использование:

```
1  floatcmp(122.45, 678.90);
```

floatcos

Вычисляет правильный косинус вещественного числа с заданной размерностью угла.

Синтаксис

floatcos(value, anglemode=radian);

Параметры функции

value – Вещественное число.
anglmode=radian – Размерность угла

Возможное использование:

```
1  new Float:fCos = floatcos(122.45, radian);
```

floatdiv

Делит вещественное число на значение, указанное вещественным числом, делителем.

Синтаксис

floatdiv(dividend, divisor);

Параметры функции

dividend –
divisor -

Возможное использование:

```
1  new Float:fDivide = floatdiv(122.45, 678.90);
```

floatfract

Вычисляет и возвращает дробную часть вещественного числа.

Синтаксис

floatfract (value);

Параметры функции

value – Вещественное число.

Возможное использование:

```
1  new Float:fFract = floatfract(122.32);
```

floatlog

Используйте эту функцию, если хотите узнать логарифм вещественного числа.

Синтаксис

floatlog(value, base=10.0);

Параметры функции

value – Вещественное число
base=10.0 – Степень логарифма.

Возможное использование:

```
1  new Float:fLog = floatlog(128.0);
```

floatmul

Перемножает два вещественных числа и возвращает произведение.

Синтаксис

floatmul(oper1, oper2);

Параметры функции

oper1 – Первое вещественное число.
oper2 – Второе вещественное число.

Возможное использование:

```
1  new Float:fMul =  floatmul(128.1, 7.9);
```

floatpower

Возводит вещественное число в степень и возвращает результат возведения числа в степень.

Синтаксис

floatpower(value, exponent);

Параметры функции

value – Возводимое вещественное число.
exponent – Степень возведения в виде вещественного числа.

Возможное использование:

```
1  new Float:fPower =  floatpower(128.1, 8.0); //1024
```

floatround

Округляет вещественное число указанным методом.

Синтаксис

floatpower(value, :method=floatround_round);

Параметры функции

value – Вещественное число.
:method=floatround_round – Метод округления, который вы хотите использовать.

Возможное использование:

```
1  new round = floatround(128.9, floatround_floor);
```

floatsin

Вычисляет синус данного вещественного числа, заданного размерностью в радианах, градусах или градиентах.

Синтаксис

floatsin (value, mode=radian);

Параметры функции

value – Вещественное число.
mode=radian – Размерность угла

Возможное использование:

```
1  new Float:fSin =  floatsin(82.4);
```

floatsqroot

Вычисляет квадратный корень данного вещественного числа и возвращает результат выражения.

Синтаксис

floatsqroot (value);

Параметры функции
value – Вещественное число.
Возможное использование:

```
1  new Float:fSin =  floatsqroot(743.34);
```

floatsub

Уменьшает первое число oper1 на число, указанное в oper2 и возвращает результат.

Синтаксис
floatsub(oper1, oper2);
Параметры функции
oper1 – Вещественное число, которое вы хотите уменьшить на заданное второе. oper2 – Значение, на которое надо уменьшить первое число.
Возможное использование:

```
1  new Float:fSub =  floatsub(128.1, 7.9);
```

floatstr

Конвертирует строку в соответствующее вещественное число.

Синтаксис
floattan (const string[]);
Параметры функции
const string[] – Строка, которую вы хотите конвертировать в вещественное число.
Возможное использование:

```
1  new Float:fFloat =  floatstr("82.4");
```

floattan

Вычисляет синус данного вещественного числа, заданного размерностью в радианах, градусах или градиентах.

Синтаксис
floattan (value, mode=radian);
Параметры функции
value – Вещественное число. mode=radian – Размерность угла
Возможное использование:

```
1  new Float:fTan =  floattan(82.4);
```

Функции STRING.INC

Основные функции для работы со строками

strlen

Возвращает количество символов в строке.

Синтаксис

strlen(const string[]);

Параметры функции

const string[] – Строковая переменная, у которой узнается количество символов

Возможное использование:

```
1  new test[64];
2  If(!strlen(test)){ //Если в строке test 0 символов
```

strval

Конвертирует строковой тип в числовой.

Синтаксис

strval(const string[]);

Параметры функции

const string[] – строковая переменная значение которой конвертируется в числовое.

Возможное использование:

```
1  new strvalue[6] = “122.23”;
2  new value = strval(strvalue);
```

strdel

Удаляет часть строки.

Синтаксис

strdel(string[], start, end);

Параметры функции

test – Строковая переменная, у которой узнается количество символов
start – символ, с которого начинается удаление
end – символ после которого заканчивается удаление

Возможное использование:

```
1  new test[64] = "Данная переменная используется для...";
2  strdel(test, 0, 17); // У нас останется  используется для...
```

strmid

Извлекает диапазон символов из строки.

Синтаксис

strmid(dest[], const source[], start, end, maxlength=sizeof dest);

Параметры функции

dest[] – строковая переменная, в которую вставляется извлеченная часть
const source[] – строковая переменная, из которой извлекается часть строки

start – символ с которого начинается извлечение
end – символ после которого заканчивается извлечение
maxlenght = sizeof dest – размер переменной, в которую вставляется извлеченная часть

Возможное использование:

```
strmid(test, "American Psycho", 10, 15); // Мы извлекли Psycho
```

strpack

Эта функция может быть использована для запаковки строки.

Синтаксис

strpack(dest[], const source[], maxlength=sizeof dest);

Параметры функции

dest[] – строковая переменная, в которую будет запакована строка.
const source[] – строка, которая будет запакована.
maxlenght = sizeof dest – размер строковой переменной, в которую будет запакована строка.

Возможное использование:

```
if(strcmp(cmdtext, "/strpack", true) == 0)
{
new message[128];
strpack(message, "Здарова чувак!");
SendClientMessage(playerid, COLOR_GREY, message);
return 1;
}
```

strunpack

Эта функция распаковывает запакованную строку в строку-назначение.

Синтаксис

strunpack(dest[], const source[], maxlength=sizeof dest);

Параметры функции

dest[] – Строка-назначение для распакованной строки.
const source[] – Текущая запакованная строка, которую нужно распаковать.
maxlenght = sizeof dest – Длина строки-назначения.

Возможное использование:

```
strunpack(string, packedString);
```

strfind

Ищет последовательность в другой последовательности.

Синтаксис

strfind(const string[], const sub[], bool:ignorecase=false, pos=0);

Параметры функции

const string[] - Строка, в которой будет производиться поиск (haystack).
const sub[] - Искомая строка (needle).
ignorecase=false - Игнорировать заглавные буквы, если true.
pos=0 - Смещение для начала поиска.

Возможное использование:

```
new instring = strfind("Are you in here?", "you", true);
```

strcmp

Сравнивает две последовательности.

Синтаксис

strcmp(const string1[], const string2[], bool:ignorecase=false, length=cellmax);

Параметры функции

const string1[] – первая последовательность
const string2[] – вторая последовательность
bool:ignorecase=false – (true/false – регистр не учитывается/учитывается).

Возможное использование:

```
1  if(strcmp(cmdtext, "/strpack", true) == 0)
```

strcat

Связывает несколько последовательностей в одну переменную.

Синтаксис

strcat(dest[], const source[], maxlength=sizeof dest);

Параметры функции

dest[] – строковая переменная, в которую вставляется последовательность
const source[] – последовательность, которая будет вставлена в переменную

Возможное использование:

```
1  new teststring[128];
2  strcat(teststring, "First string\n");
3  strcat(teststring, "Second string\n");
4  strcat(teststring, "Third string");
5  SendClientMessage(playerid, COLOR_RED, teststring);
```

valstr

Конвертирует целое число в строку.

Синтаксис

valstr(dest[], value, bool:pack=false);

Параметры функции

dest[] – строковая переменная в которую вставляется последовательность
value - Конвертируемое целое число.
pack=false - Запакует строку, если true.

Возможное использование:

```
1  valstr(string, 454);
```