fastman92 limit adjuster documentation



The following document attempts to describe the functionality offered by the project “fastman92 limit adjuster”.

Author: **fastman92**

Website: <http://fastman92.com>

# Table of contents

Contents

[Table of contents 2](#_Toc32769409)

[What is fastman92 limit adjuster? 7](#_Toc32769410)

[How does the FLA work? 8](#_Toc32769411)

[How to check, which features are supported? 9](#_Toc32769412)

[Use the FLA for games on Android 9](#_Toc32769413)

[How libraries will be modified? 9](#_Toc32769414)

[Quick review of steps in ‘Android plugin deployment solution’ application 10](#_Toc32769415)

[How to install the FLA? 10](#_Toc32769416)

[Unpack the archive, which the FLA got shipped in. 10](#_Toc32769417)

[Run Android plugin deployment solution 10](#_Toc32769418)

[Option “Exit script immediately” 11](#_Toc32769419)

[Select game and game version 11](#_Toc32769420)

[Read original game library 12](#_Toc32769421)

[Write new libraries 17](#_Toc32769422)

[How to access the external file directory for specified game? 27](#_Toc32769423)

[Access it on the computer over network 28](#_Toc32769424)

[Access it on the Android device 31](#_Toc32769425)

[How to establish the ADB connection between computer and Android device? 32](#_Toc32769426)

[Two ways to establish the ADB connection 32](#_Toc32769427)

[Make connection by USB 32](#_Toc32769428)

[Make connection by network 32](#_Toc32769429)

[Check if ADB connection is working 33](#_Toc32769430)

[Monitor debug output 33](#_Toc32769431)

[How to create a plugin for game on Android? 35](#_Toc32769432)

[Introduction 35](#_Toc32769433)

[Setting up the environment variables 36](#_Toc32769434)

[Open the sample project and start working 36](#_Toc32769435)

[Install a custom plugin 38](#_Toc32769436)

[How to test the newly installed plugin 39](#_Toc32769437)

[How plugin code is executed? 41](#_Toc32769438)

[OnPluginLoad 42](#_Toc32769439)

[OnApplicationInit 46](#_Toc32769440)

[Debugging 49](#_Toc32769441)

[How to start the IDA Pro server? 49](#_Toc32769442)

[How to debug the game? 49](#_Toc32769443)

[How to debug the game, when process is starting? 49](#_Toc32769444)

[Features offered by the FLA 50](#_Toc32769445)

[DYNAMIC LIMITS 50](#_Toc32769446)

[ColModels 50](#_Toc32769447)

[VehicleStructs 51](#_Toc32769448)

[rwObjectInstances 51](#_Toc32769449)

[Matrices 52](#_Toc32769450)

[PtrNode Singles 52](#_Toc32769451)

[PtrNode Doubles 52](#_Toc32769452)

[PtrNodes 52](#_Toc32769453)

[EntryInfoNodes 52](#_Toc32769454)

[Peds 52](#_Toc32769455)

[Vehicles 52](#_Toc32769456)

[Objects 53](#_Toc32769457)

[Tasks 53](#_Toc32769458)

[CDummyTask 53](#_Toc32769459)

[Events 53](#_Toc32769460)

[PointRoute 53](#_Toc32769461)

[PatrolRoute 53](#_Toc32769462)

[InteriorInst 53](#_Toc32769463)

[PortalInst 53](#_Toc32769464)

[CAnimBlender 53](#_Toc32769465)

[CAtdNodeAnimChangePooledObject 53](#_Toc32769466)

[CAtdNodeAnimPlayer 53](#_Toc32769467)

[crFrameFilterBoneMask 53](#_Toc32769468)

[crFrameFilterBoneAnalogue 53](#_Toc32769469)

[crExpressionProcessor 53](#_Toc32769470)

[crmtObserver 53](#_Toc32769471)

[NavMeshRoute 53](#_Toc32769472)

[Cam 53](#_Toc32769473)

[ExplosionType 53](#_Toc32769474)

[NodeRoute 53](#_Toc32769475)

[TaskAllocator 53](#_Toc32769476)

[PedIntelligence 53](#_Toc32769477)

[PedAttractors 53](#_Toc32769478)

[QuadTreeNodes 53](#_Toc32769479)

[Collision links 53](#_Toc32769480)

[CustomEnvMapPipeMatDataPool 53](#_Toc32769481)

[CustomEnvMapPipeAtmDataPool 53](#_Toc32769482)

[CustomSpecMapPipeMaterialDataPool 53](#_Toc32769483)

[IPL 53](#_Toc32769484)

[Buildings 53](#_Toc32769485)

[Dummies 54](#_Toc32769486)

[Inst entries per file 54](#_Toc32769487)

[Entity index array 54](#_Toc32769488)

[Map zones 54](#_Toc32769489)

[Navigation zones 54](#_Toc32769490)

[COccluder, apply coordinate limit patch 54](#_Toc32769491)

[MAP LIMITS 54](#_Toc32769492)

[Enable track config loader 54](#_Toc32769493)

[Enable path debugging 56](#_Toc32769494)

[Paths map size 56](#_Toc32769495)

[Radar map size 56](#_Toc32769496)

[Enable frontend map different 57](#_Toc32769497)

[World map size 57](#_Toc32769498)

[LEVEL LIMITS 57](#_Toc32769499)

[VEHICLE SPECIAL FEATURES 57](#_Toc32769500)

[IMG LIMITS 58](#_Toc32769501)

[Enable handling of new enhanced IMG archives 58](#_Toc32769502)

[Enable handling of IMGLIST keyword 58](#_Toc32769503)

[ID LIMITS 59](#_Toc32769504)

[Apply ID limit patch 61](#_Toc32769505)

[HANDLING.CFG LIMITS 61](#_Toc32769506)

[OTHER LIMITS 62](#_Toc32769507)

[LOD distance 62](#_Toc32769508)

[Number of plate textures 62](#_Toc32769509)

[Coronas 63](#_Toc32769510)

[ROADBLOCK LIMITS 63](#_Toc32769511)

[Apply roadblox.dat better loader 63](#_Toc32769512)

[ADDONS 63](#_Toc32769513)

[Cheat String Loader 63](#_Toc32769514)

[Vehicle audio loader 64](#_Toc32769515)

[FXT loader 65](#_Toc32769516)

[Radar blip sprite filename loader 66](#_Toc32769517)

[Train type carriages loader 67](#_Toc32769518)

[Model special features loader 68](#_Toc32769519)

[Not a prop ID range loader 70](#_Toc32769520)

[MAIN 70](#_Toc32769521)

[Use a different INI 70](#_Toc32769522)

[CLEO for Android 70](#_Toc32769523)

[Enhanced IMG format 71](#_Toc32769524)

[How to make archives in enhanced IMG format? 71](#_Toc32769525)

[Test of archive: 72](#_Toc32769526)

[Structure of IMG archive. 72](#_Toc32769527)

[Path format 73](#_Toc32769528)

[1. Header 73](#_Toc32769529)

[The first part 74](#_Toc32769530)

[The second part: 74](#_Toc32769531)

[The numbers of elements 74](#_Toc32769532)

[2. Path nodes 75](#_Toc32769533)

[3. Car path links 77](#_Toc32769534)

[5. Adjacent path node addresses 77](#_Toc32769535)

[6. Adjacent car path link addresses 77](#_Toc32769536)

[7. Distance to adjacent path node addresses 78](#_Toc32769537)

[8. Path Intersection Flags 78](#_Toc32769538)

[9. EOF 79](#_Toc32769539)

[How to make projects compatible with the FLA? 79](#_Toc32769540)

[Integrate ForOtherProjects into your project 79](#_Toc32769541)

[Use FLA\_not\_required 80](#_Toc32769542)

[Use FLA\_required 80](#_Toc32769543)

[ID limits 82](#_Toc32769544)

[Table of possible IDs 82](#_Toc32769545)

[CFileID 83](#_Toc32769546)

[*How to integrate CFileID into structures* 83](#_Toc32769547)

[List of structures containing 16-bit model ID 85](#_Toc32769548)

[List of structures containing 16-bit TXD ID 87](#_Toc32769549)

[File info streaming 87](#_Toc32769550)

[Individual types 89](#_Toc32769551)

[uint8\_replacement 90](#_Toc32769552)

[Shadow limits 90](#_Toc32769553)

[Real time shadows 90](#_Toc32769554)

[Number of elements in m\_realTimeShadowPool from CRealTimeShadowManager 91](#_Toc32769555)

[Weapon limits 91](#_Toc32769556)

[Extended weapon ID 91](#_Toc32769557)

[Parent weapon ID 92](#_Toc32769558)

[Weapon name 92](#_Toc32769559)

[Count of weapon infos 92](#_Toc32769560)

[Number of weapon types 92](#_Toc32769561)

[Vehicle colors 92](#_Toc32769562)

[Reallocated color array 92](#_Toc32769563)

[Extended weapon ID 93](#_Toc32769564)

[Adding vehicles to GTA / Bully 94](#_Toc32769565)

[Adding weapons to GTA 97](#_Toc32769566)

[Explanation of world sectors 97](#_Toc32769567)

[How map is divided into sectors 97](#_Toc32769568)

[World sectors 98](#_Toc32769569)

[World repeat sectors 98](#_Toc32769570)

[World LOD sectors 99](#_Toc32769571)

# What is fastman92 limit adjuster?

fastman92 limit adjuster is the project, which adds many possibilities to specific games. In this document, the term FLA will be used – which denotes the FLA.

Before the FLA was made the game modding was severely limited – it was possible to add content to games, but only to a limited extent. With the FLA, the possibilities have been greatly increased - it’s possible to make very big maps, alter the coordinate limits, add lots of vehicles, increase the renderer or visibility limits and bring other functionality.

# Note that not all functionality is supported for all games in the FLA. The support varies by game.

# How does the FLA work?

The FLA is the library, which is loaded under the process, before the application’s code gets to execute.

On the Windows operating systems, the FLA is actually the DLL library with the extension renamed to ASI, which the ASI loaders can load. The proper ASI loader should load the FLA before any other ASI plugin – because of the FLA’s filename starting with “$” character, which should make the FLA first on list of all ASI plugins, when sorted alphabetically.

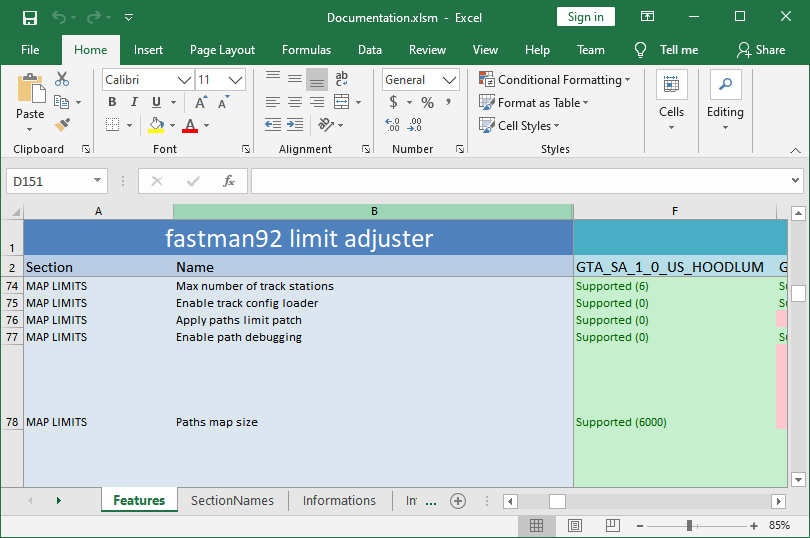
The FLA internally modifies a huge number of instructions of the existing application executable code. For that a proprietary technology, which allows to create massive patches in partly automated way and which targets a problem of scalability is used. This is to make sure as much as possible that the patches will be free of any errors, unless the person creating them has forgotten something or made a logical error.

Used correctly, it allows to make huge and very complex patches, that would have never been possible before (speaking of the GTA community).

# How to check, which features are supported?

Different games and their game versions have a different set of features supported.

So how do we find out which features are supported for specified game versions? Read on.

* Firstly, if the INI file does not exist, then it gets generated with options that are supported by the game, from which the FLA got started.
* Secondly, it’s a good idea to see **Documentation.xmlx**, the file from the FLA archive, which you’ve got. This file contains a table, which tells what features are supported.  
  Let’s take a look:  
  

Let’s see one value from the screenshot:

Supported (6) – means the feature is implemented (the limit can be changed) and the default value for this key is 6.

# Use the FLA for games on Android

## How libraries will be modified?



## Quick review of steps in ‘Android plugin deployment solution’ application



## How to install the FLA?

There are few steps to install the FLA on game for Android.

They must be followed properly.

### Unpack the archive, which the FLA got shipped in.

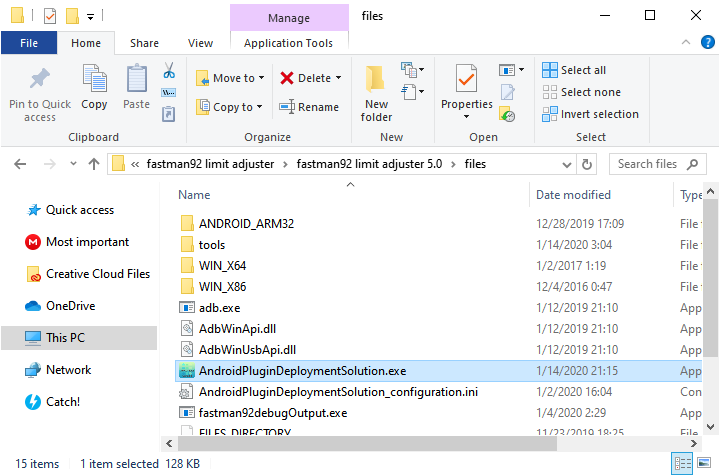
The FLA is shipped inside the archive. You need to extract all files from this archive to some directory.

### Run Android plugin deployment solution

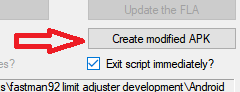
Then run **Android plugin deployment solution**, which is a small application.

Following things are required to run **Android plugin deployment solution**:

* .NET Framework 2.0 or higher, already a part of Windows 10
* Official Java – only required if you want to use an option “**Create modified APK**”



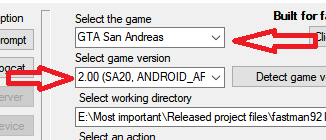
### Option “Exit script immediately”



The scrips executed from APDS get closed immediately by default.  
If you’ve got any possible problems while executing the batch scripts from ADPS, then untick the option “**Exit script immediately**” and batch scripts from command prompt will no longer close automatically.

### Select game and game version

Game and game version needs to be selected.

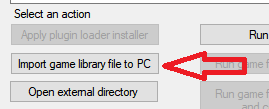


### Read original game library

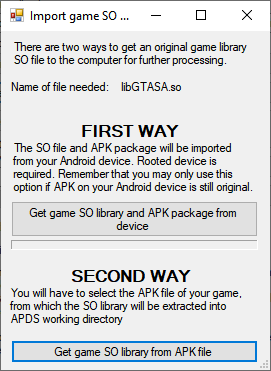
An original game library such as libGTASA.so needs to be imported for further processing. It needs to be read by Android plugin deployment solution.

*The original game library only needs to be imported once for selected game version.*

Click the button “**Import game library file to PC**”:



The following window will be shown:

  
There are two possible ways to import the original library file of game:

* *Import the original library file from your Android device, where you have the game installed.*
  + Root access is **required**.
  + [ADB connection needs to be working.](#_How_to_establish)
  + Original game library for currently selected game version needs to be installed on the device.

[Click to learn how to read the original file directly from the device.](#_Read_original_file)

* *Import the original library from APK package* 
  + Root access is **not required**.
  + You need to have original APK package of your game accessible on the computer.

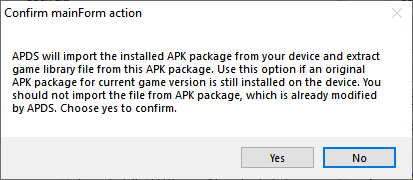
[Click to learn how to read the original library file from APK package.](#_Read_original_file_1)

#### Read original file directly from Android device

Click button “**Get game SO library from Android device**”



The following message box will be shown:

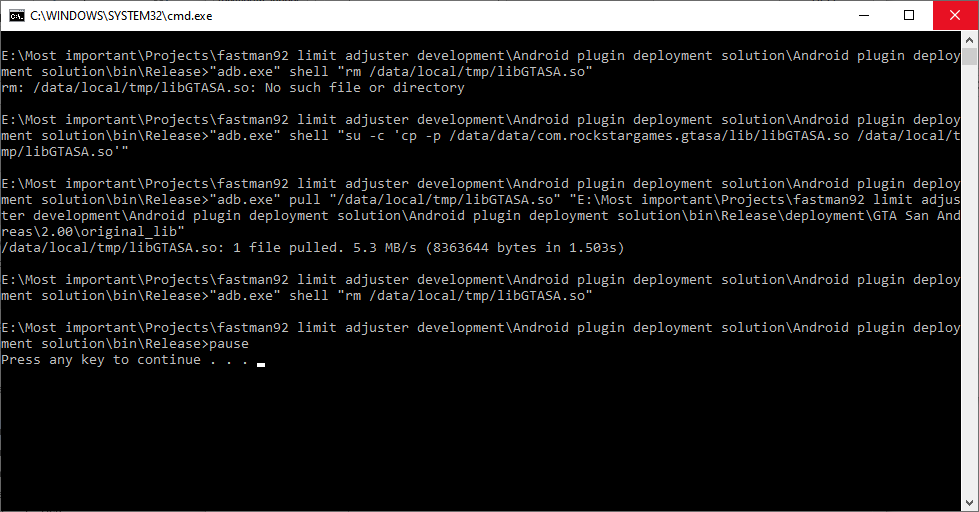


Basically, it means that you should only use this option, if the original APK package is still installed on the device for currently selected game version.

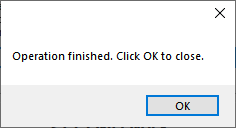
Do not write a new modified game library to the device and import it back again. It would be a mistake. Hence, this message of warning.

Click “Yes”.

The command prompt will appear for a while:

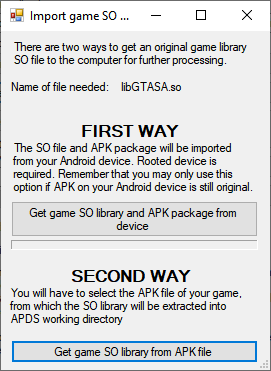


The message box will appear:

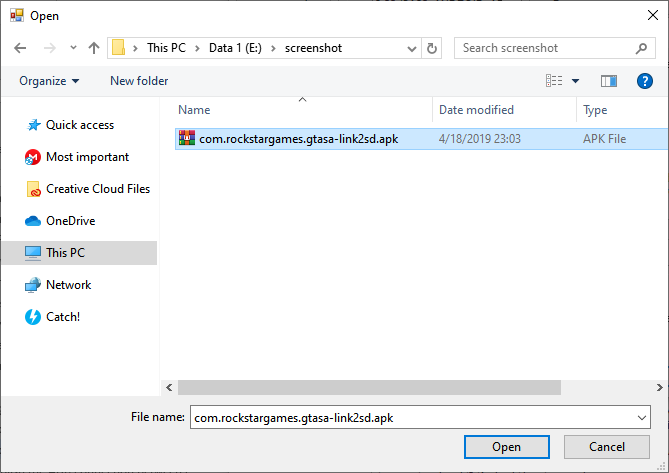


#### Read original file from APK package of game

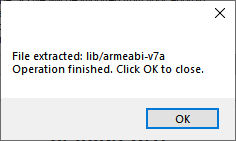
Click button “**Get game SO library from APK file**”



Select the original APK package file:



When operation completes, the following message will be shown:



### Write new libraries

When the original game library is imported, then we’re ready for the next step – writing new libraries.

There are two ways to write the new libraries:

* *Write new libraries directly on your Android*
  + Root access is **required**.
  + [ADB connection needs to be working.](#_How_to_establish)
  + It was noticed, Android may sometimes restore the original **lib** directory from **base.apk**, which effectively means the FLA libraries needs to be installed once again from time to time. This problem has been observed on Xiaomi Redmi 5 Plus. That’s a problem that still needs researching.

[Click to learn how to write new libraries directly to the device.](#_Write_new_libraries)

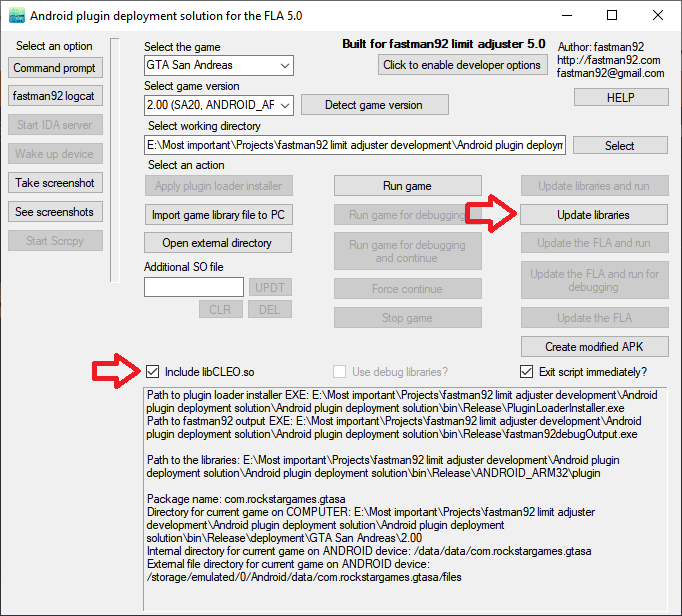
* *Import the original library from APK package* 
  + Root access is **not required**.
  + You need to have an original APK package of your game accessible on the computer.
  + New APK package will be signed with the test key. Different key from the one used by original publisher of APK package.

[Click to learn how to create a new APK package with the FLA and other libraries.](#_Read_original_file_1)

#### Write new libraries directly to Android device

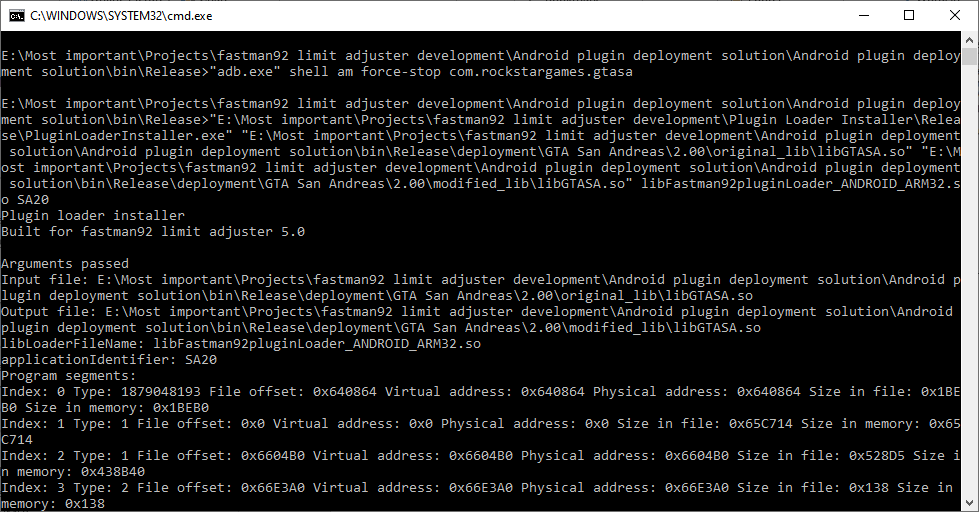
Check or uncheck the option “include libcleo.so” appropriately.

You need to decide if CLEO should be added to your game.



Then click the button “**Update libraries**”.

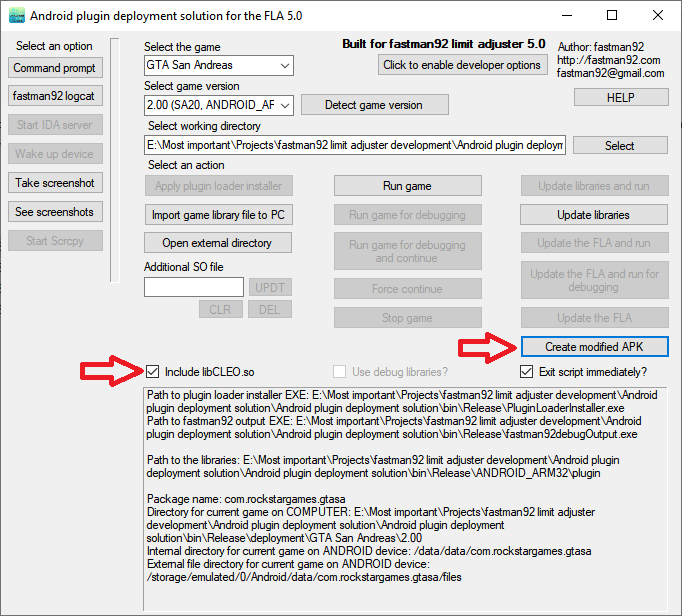
The batch script will then be executed in command prompt, wait until it finishes:



#### Create a new APK package with the FLA and other libraries

Check or uncheck the option “**include libcleo.so**” appropriately.

You need to decide if CLEO should be added to your game.

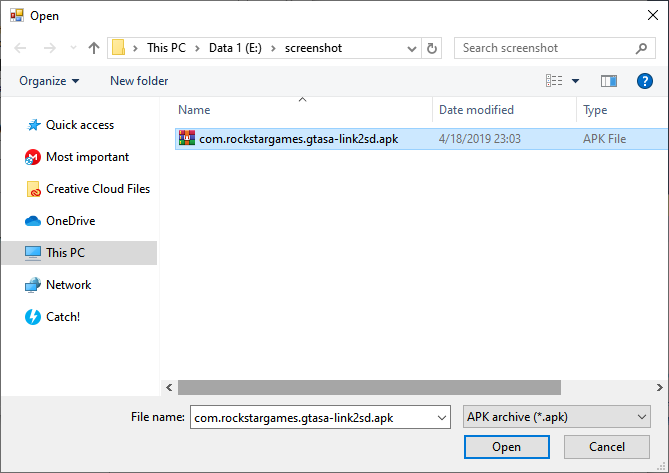


Then click the button “**Create modified APK**”.

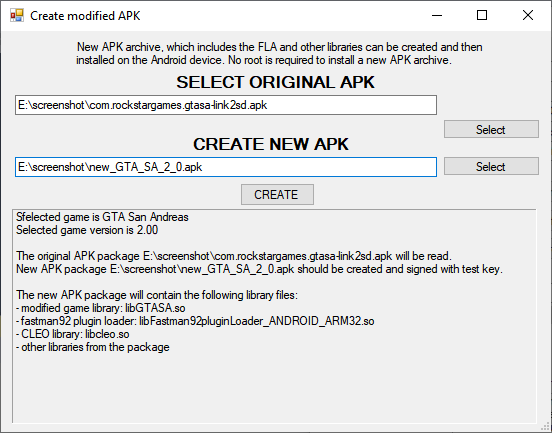
The following window will be shown:



Please select the original APK file:

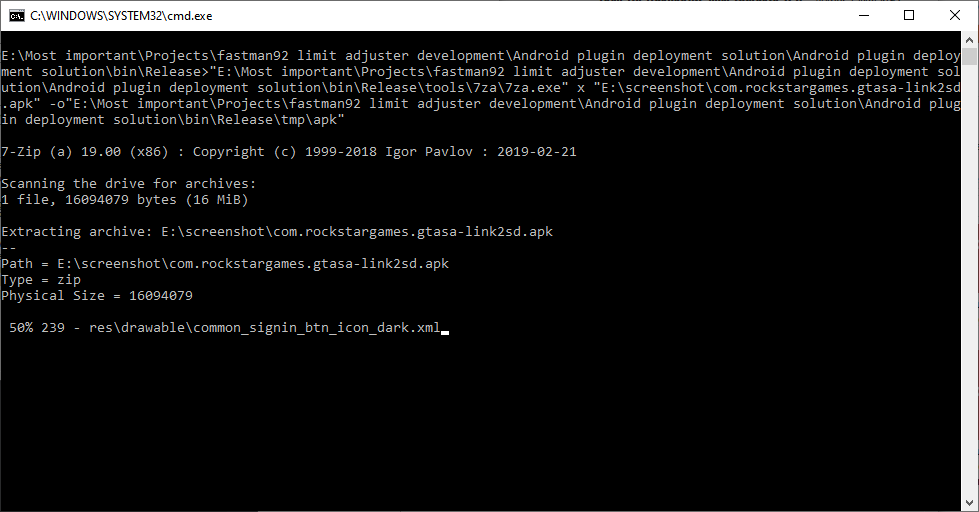


Then select the path of new APK package.



When paths are set up, click the button “**CREATE**”.

The command prompt will pop up, wait until the work is done.

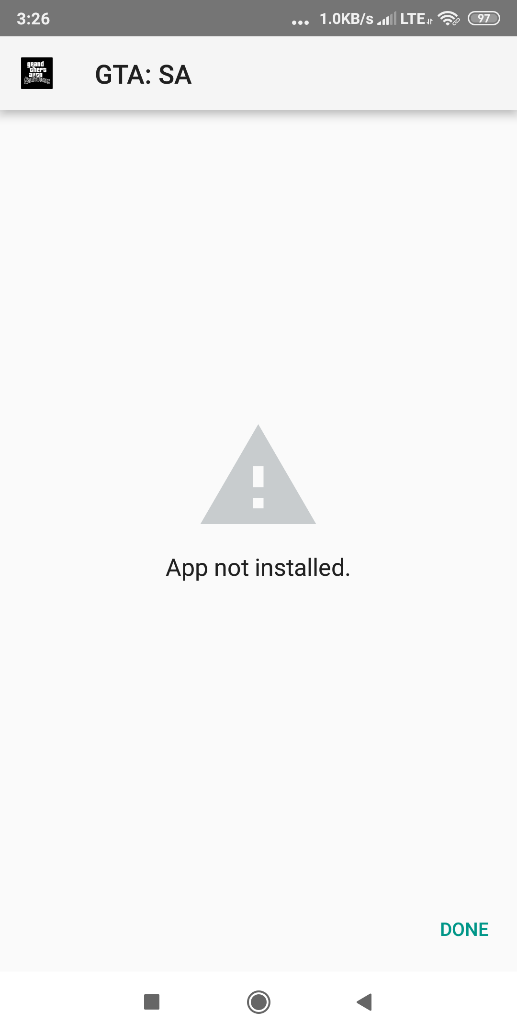


New APK package should be created and can then be moved to the device.

Transfer it from your computer to your Android device and install the new package.

**Remember that the new APK package will be signed with different key to the one that the publisher of game used originally.**

**Because of this, the new APK package will not install if game is still installed on the Android device:**

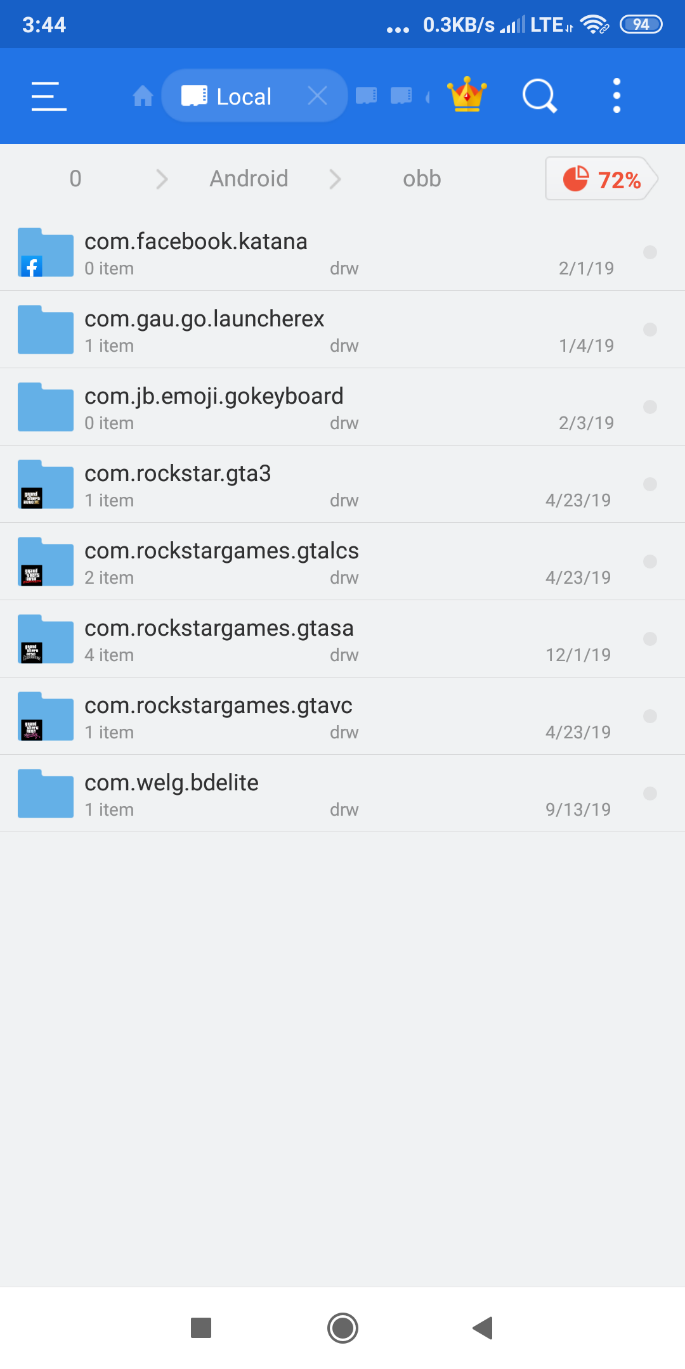


**Old package needs to be installed before the new one with different key can be installed.**  
If you’ve got such a problem please follow the steps below.

##### Uninstall the game, but keep its OBB files.

The game downloads OBB files, which are quite big. They can take a long time to download, especially on slow Internet connections. Because of this it might be a good idea to uninstall the game and still keep the OBB files.

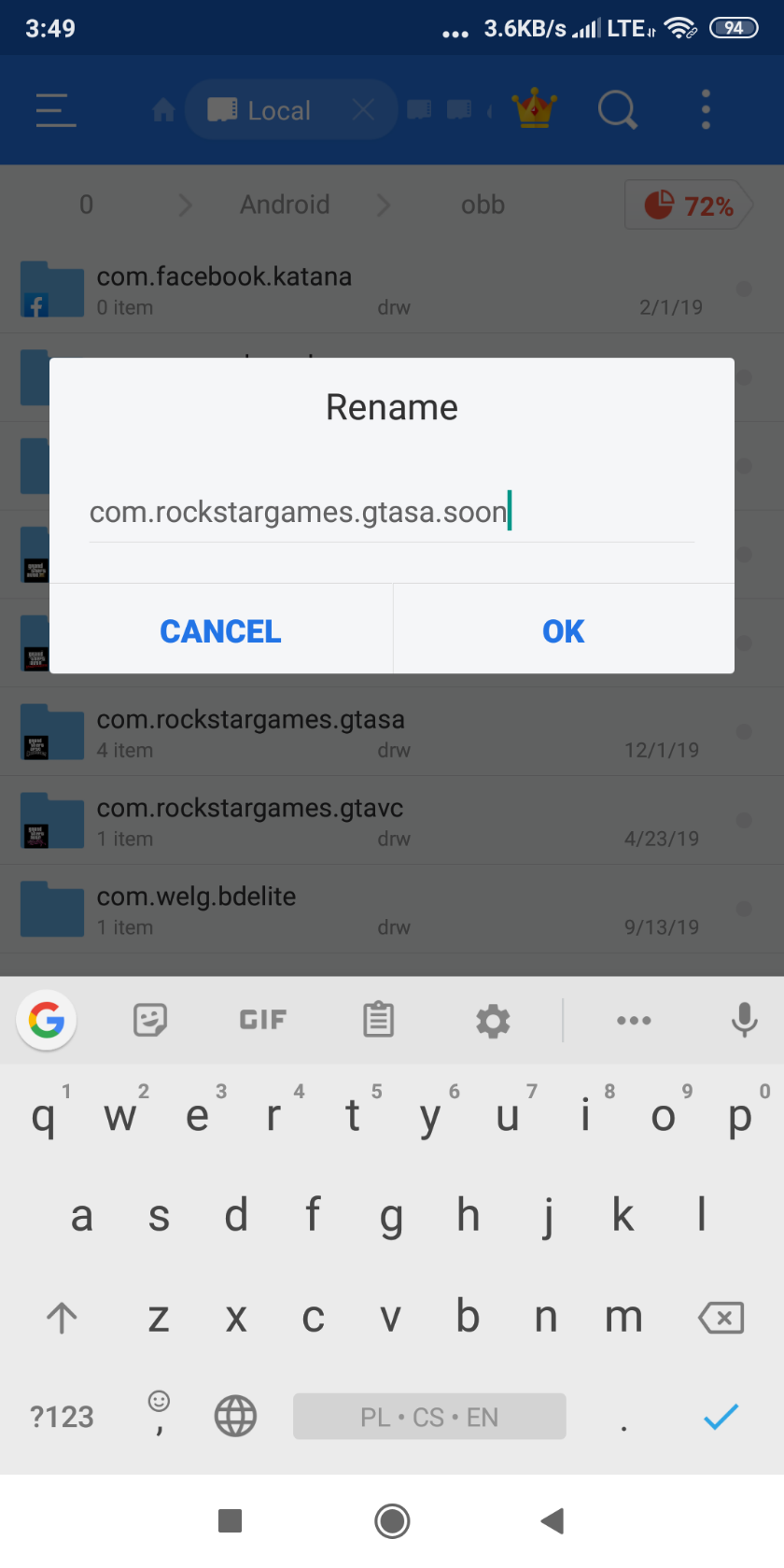
Launch your file manager (it could be ES File Explorer available in Play Store) and open the following directory: **/sdcard/obb/**



Please rename the directory of your game to something else.

For example if GTA San Andreas application needs to be uninstalled, then you can rename

**com.rockstargames.gtasa** into **com.rockstargames.gtasa.soon**



Now the game can be uninstalled without deleting its OBB files.

Uninstall the game.

Restore the previous name of OBB directory.

In the example the name of directory **com.rockstargames.gtasa** needs to be restored.

##### Install the new APK package

You can now install the APK package signed with any key.

## How to access the external file directory for specified game?

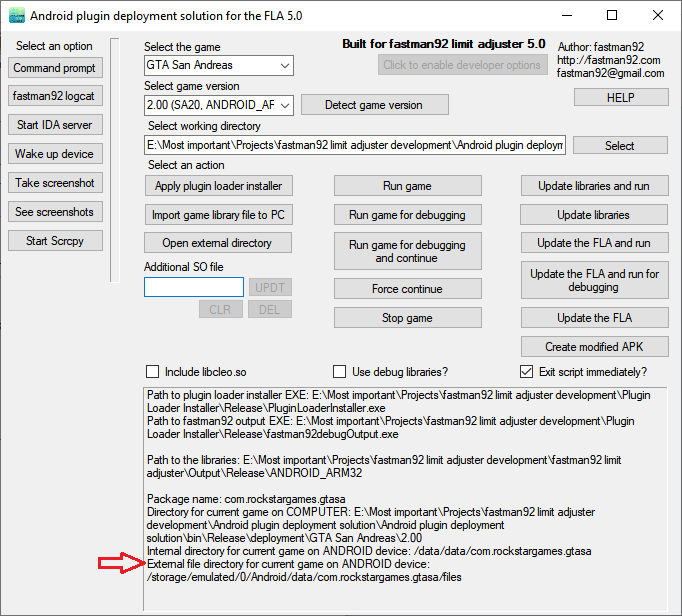
The external file directory, which the game uses to read files. This is also the directory, where main INI of the FLA will be created and where you may put additional files of the FLA.

This directory contains the following things:

* The FLA INI
* The FLA log
* The FLA additional files for example “**data\gtasa\_vehicleAudioSettings.cfg**”
* Other files that can be read by the game, for example “**data\handling.cfg**”

Please see the property “**External file directory for current game on ANDROID device**”.

Let’s take a look:



As can be seen, the external file directory for current game on the sample picture is **/storage/emulated/0/Android/data/com.rockstargames.gtasa/files/**

There are at least two ways to access this directory:

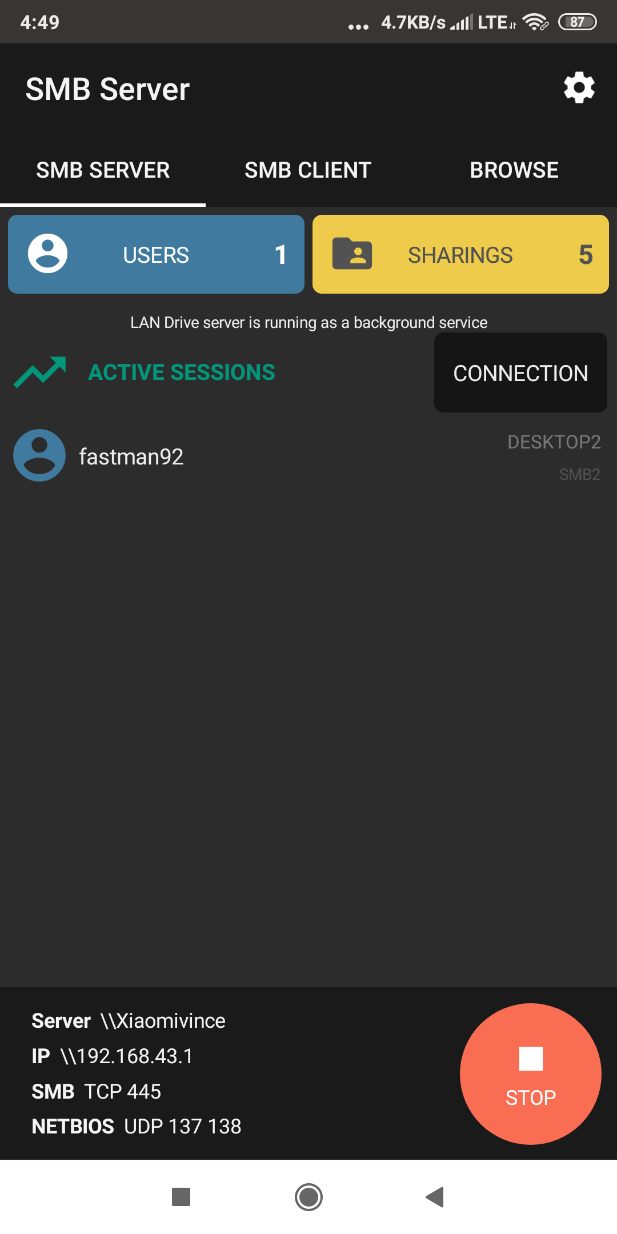
* Access it on the computer over network, works with the least problems if Android device is rooted
* Access it on the Android device

### Access it on the computer over network

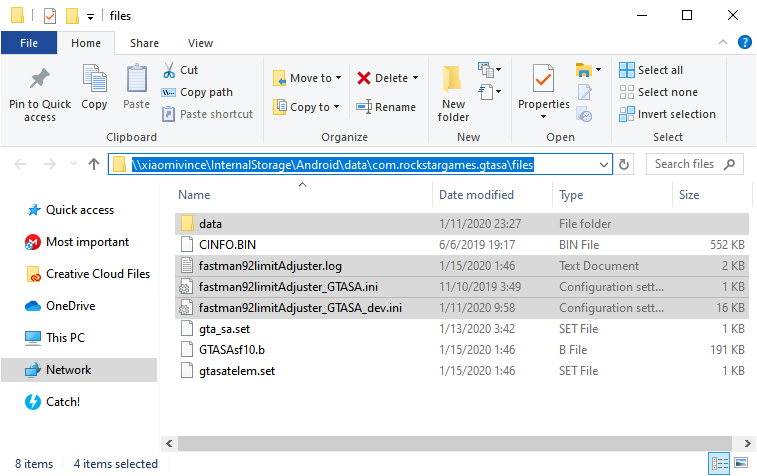
It’s possible to access the files on the computer using the network file system.

Both the computer and Android device need to stay in the same LAN network.

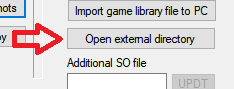
This can be accomplished with the application [LAN drive - SAMBA Server & Client](https://play.google.com/store/apps/details?id=fr.webrox.landrive)



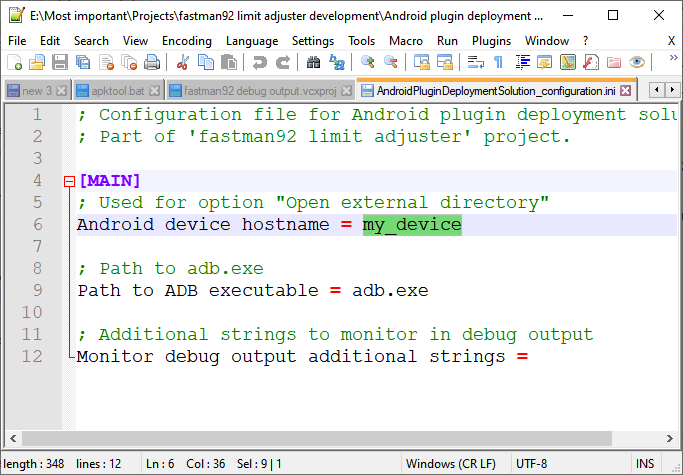
Let’s take a look:



**Android plugin deployment solution** has got a button to open the **external file directory**.

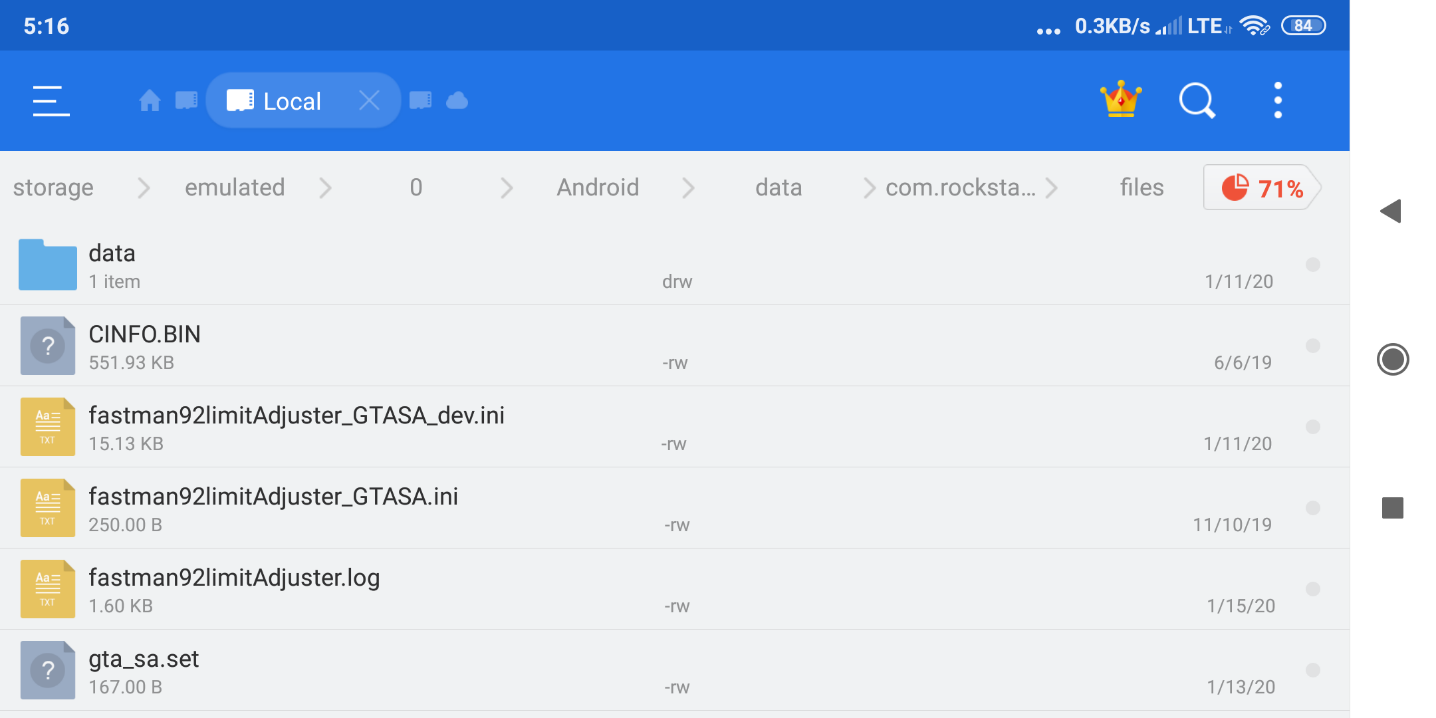


The hostname or IP address depends on the INI configuration:



### Access it on the Android device

File manager can be used. I recommend ES File Explorer:



## How to establish the ADB connection between computer and Android device?

Android Debug Bridge (adb) is a versatile command-line tool that lets you communicate with a device. The adb command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to a Unix shell that you can use to run a variety of commands on a device. It is a client-server program that includes three components:

***ADB*** *can be used to install the FLA on Android device by accessing the library files of game directly. This includes reading an original game library and writing new library files.*

### Two ways to establish the ADB connection

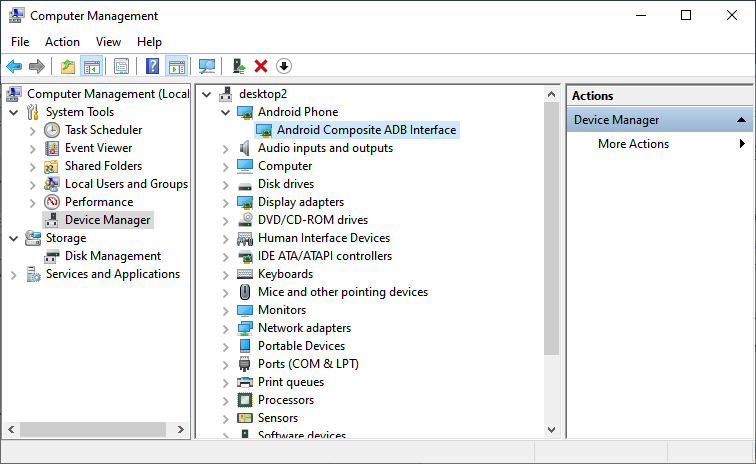
It's possible to establish the ADB connection in two ways:

* [USB connection](#_Make_connection_by)
* [Network connection](#_Make_connection_by_1) (usually Wi-Fi, some tablets may have RJ45 port as well)

Both ways are described in the document.

### Make connection by USB

Make sure that you have got ADB driver installed.  
To make connection simply connect the USB cable to your Android device.

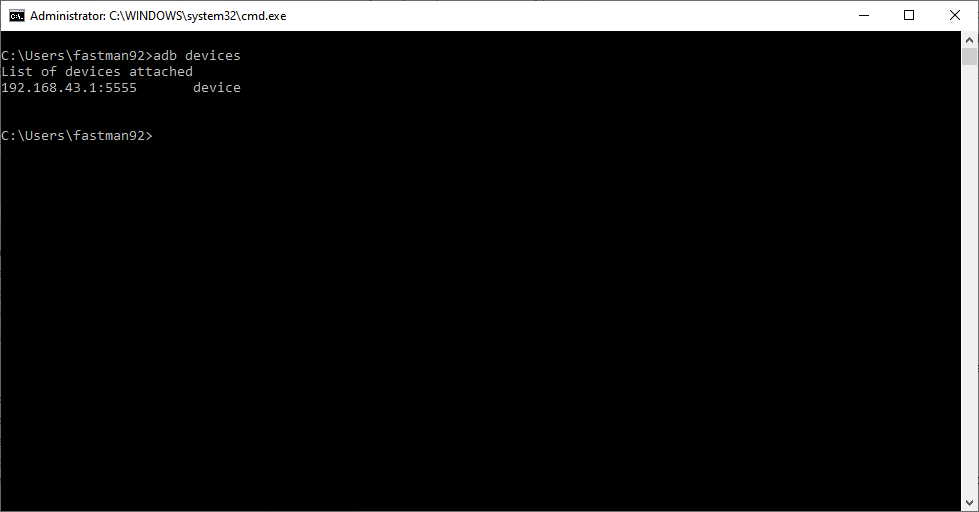


[Then check if the ADB connection is working properly.](#_Check_if_ADB)

### Make connection by network

ghf

### Check if ADB connection is working

Click “Command prompt” in ‘Android plugin deployment solution’.  
Type: *adb devices*  
to get an information. You should have only one ADB device on the list.  


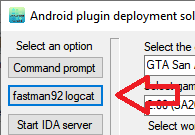
## Monitor debug output

**Android plugin deployment solution** has got an application, which is used to monitor the following things:

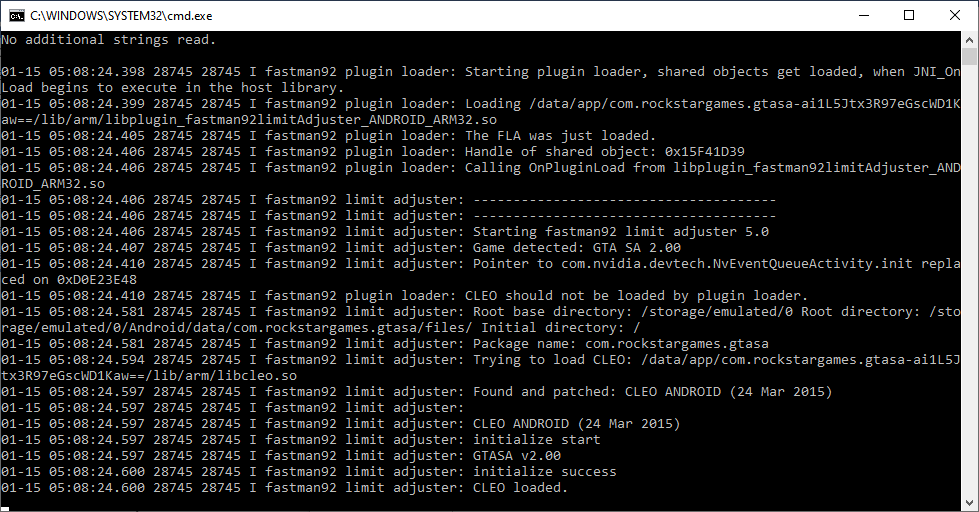
* fastman92 plugin loader
* fastman92 loader
* additional strings can be monitored

Moreover, the console output is cleared, when the plugin loader (when the game starts), so there are no multiple lines of consecutive runs of the game.

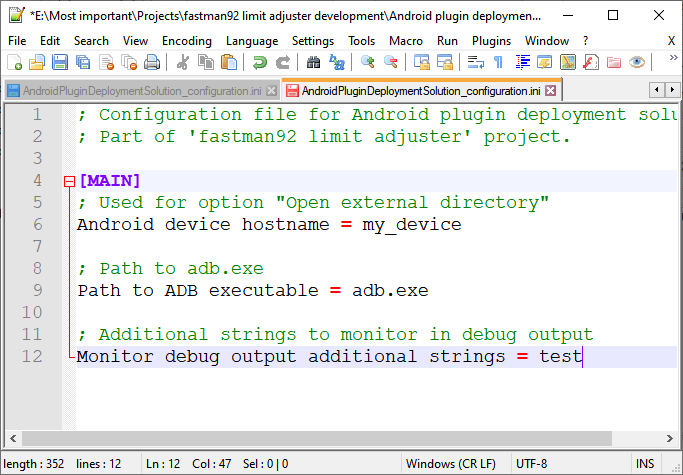
The debug output may be viewed by clicking the button “**fastman92 logcat**”:



Let’s see:



The application also supports additional strings, which cause the lines to be printed if containing these additional strings. These can be defined by option “Debug output additional strings” in the INI file:



# How to create a plugin for game on Android?

## Introduction

It’s possible to build the plugins for games on Android, which can be then be loaded by the plugin loaded.

Important things to know

* The device should be rooted for a quick development, otherwise the only way to test the plugin is building a new APK package by ‘Android plugin deployment solution’ application, sending it on the device in some way and installing the APK on the device. Impractical.
* Click the button **Click to enable developer options** in ‘Android plugin deployment solution’ to enable the options, which standard users do not need and which are necessary for the development of plugins.
* the filename of plugin needs to match this pattern: **libplugin\*.so**
* Current ARM processors support the following modes:
  + Thumb – instructions take 2 or 4 bytes.
  + ARM – every instruction takes 4 bytes.
* Jump pointers / function pointer
  + The target encoding is written as a part of address itself, in the lowest bit.
  + If lowest bit has value **1**, then the address points into Thumb instruction
  + If lowest bit has value **0**, then the address points into ARM instruction.
  + As an example, jumping to address 0x57025 would make the CPU execute the instructions from 0x57024 in Thumb mode.
  + Now jumping to address 0x56024 would make the CPU execute the instructions from 0x57024 in ARM mode.
* How many bytes are necessary to make a branch to any address?
  + Thumb
    - If source address is aligned to value of 4, then trampoline takes 8 bytes.
    - If source address is **NOT** aligned to value of 4, then trampoline takes 10 bytes
  + ARM – 8 bytes
* Compile the project in Release|ARM mode in Visual Studio.
* Use “fastman92 logcat” from ‘Android plugin deployment solution’ application to see the debug output.
* Plugin loader has got some functions exported. See Exports.h from the plugin loader.
* Use root directory from GetStorageRootDirectoryPath to read/write the game files.
* Plugins have the following stages:
  + OnPluginLoad – when the plugin library loaded
  + OnApplicationInit – when the code of game/application has done initialization. Paths are already set up at this stage.

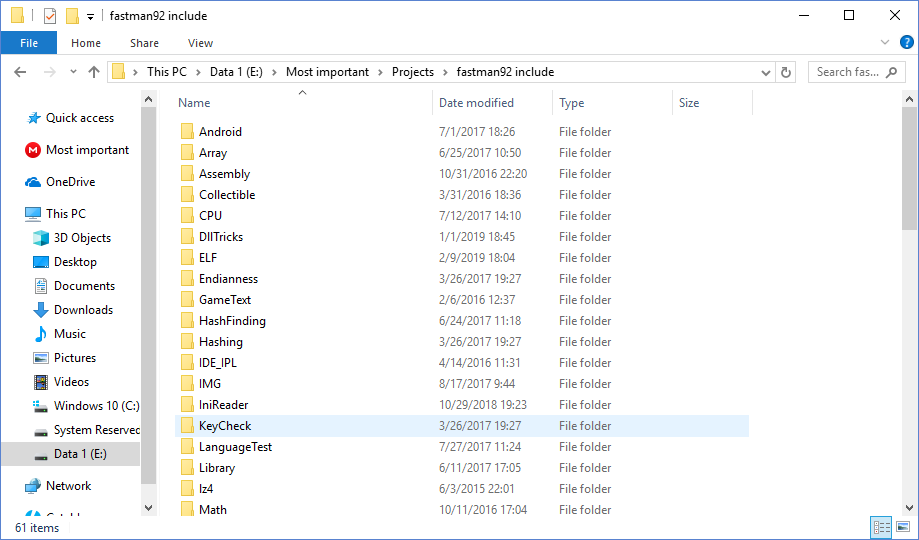
Requirements:

* **Visual Studio with Android NDK** support needs to be installed on the computer to develop plugins.
* **Java Development Kit** (jdb.exe) is not necessary, but can be useful, if you want to do the debugging of your plugin or game. The buttons ‘Run application for debugging and continue’ and ‘Force continue’ in ‘Application plugin deployment deployment solution’ rely on the **JDK** being installed and the path to jdb.exe from **JDK** is retrieved from the registry.

## Setting up the environment variables

System variable **FASTMAN92\_INCLUDE** needs to be set up, it will be used later in the project.

The directory with files, that can be included in C++ project.



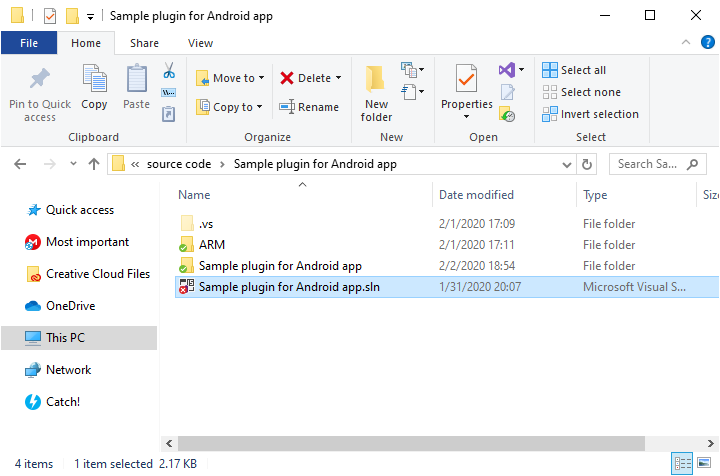
Set it appropriately to the path of unpacked “fastman92 include” from archive, which included the FLA.

System variable **FASTMAN92\_INCLUDE** will be used to hold a path to this directory.

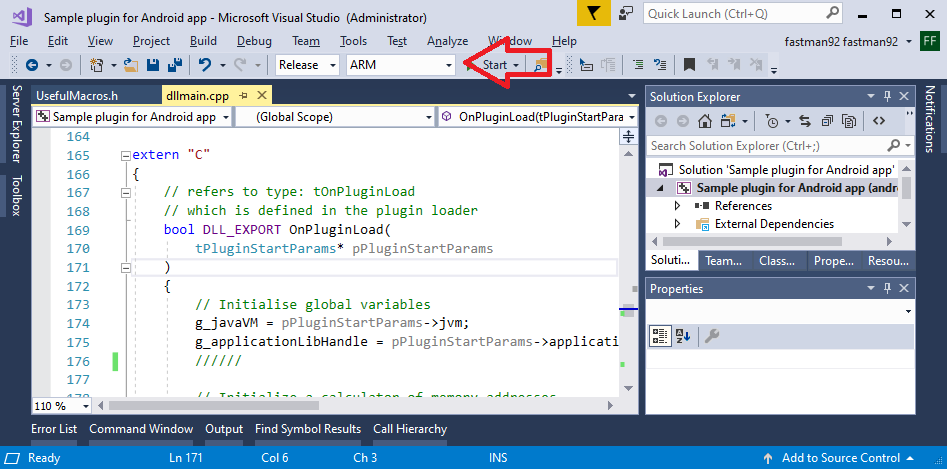
## Open the sample project and start working

The FLA provides a sample project, which can be used to start the development of your plugin.

You can open the **SLN** file.

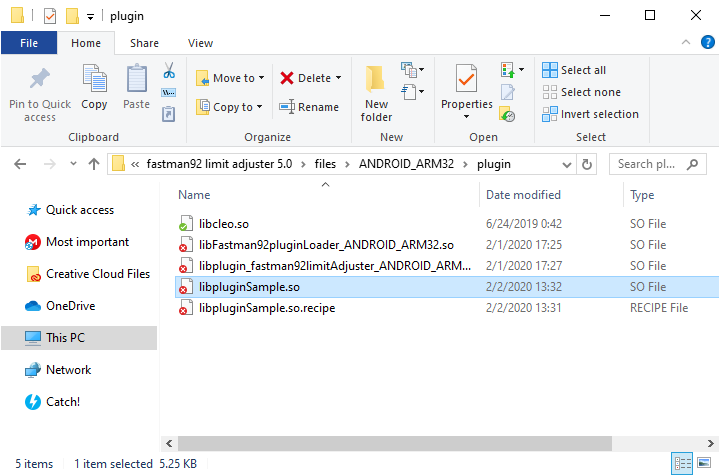


Set the solution configuration to **Release** and solution platform to **ARM**.



Now you can try compiling.

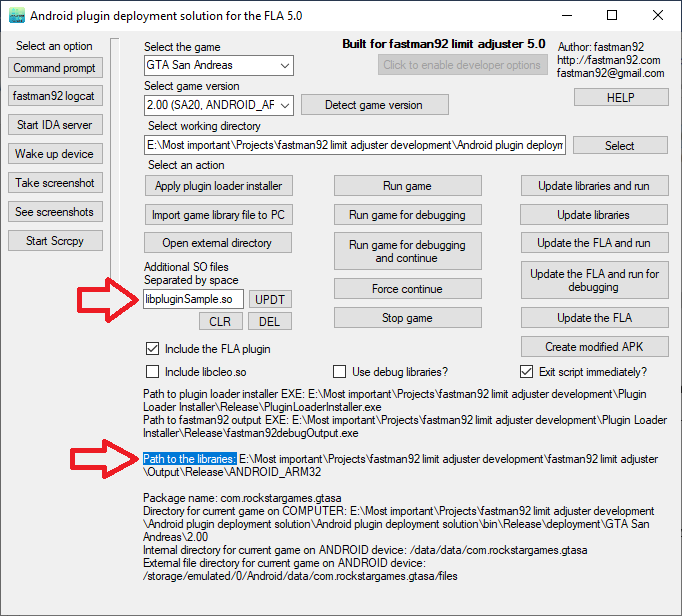
If successful, you should get a file in the directory appropriate for plugins, the directory “Path to the libraries” in the application’ Android plugin deployment solution.’



## Install a custom plugin

The file of new plugin should be in the appropriate directory, known as “**Path to the libraries**”.

‘Android plugin deployment solution’ allows to install custom library files from this directory.



If you still haven’t done it, [set the correct game and game version](#_Select_game_and).

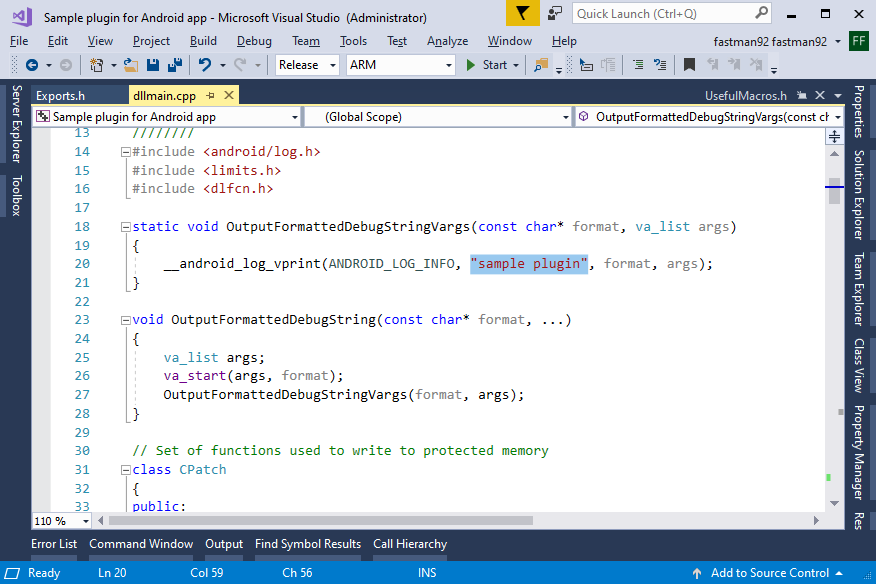
Then set the of additional SO file like on the picture.

Click **UPDT** button to install the SO library into game.

## How to test the newly installed plugin

The plugin may produce a debug output.

Our sample plugin will produce debug strings with tag such as “sample plugin”:

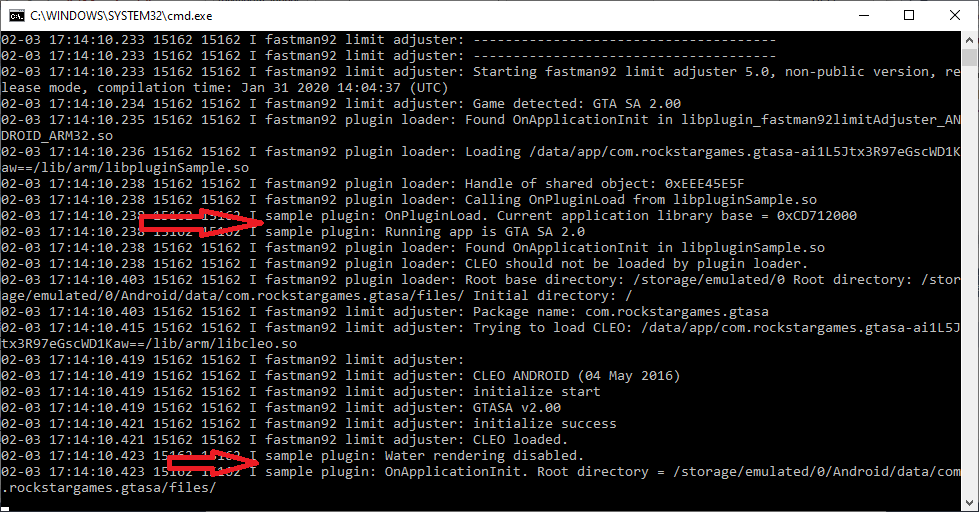


Read the chapter, which explains [how to monitor the debug output.](#_Monitor_debug_output)

Sample plugin will disable the water rendering (CWaterLevel::Render function) on GTA SA 2.0 on Android.

Click **Run game** to run the game.

If sample plugin is installed, you should see the following lines:



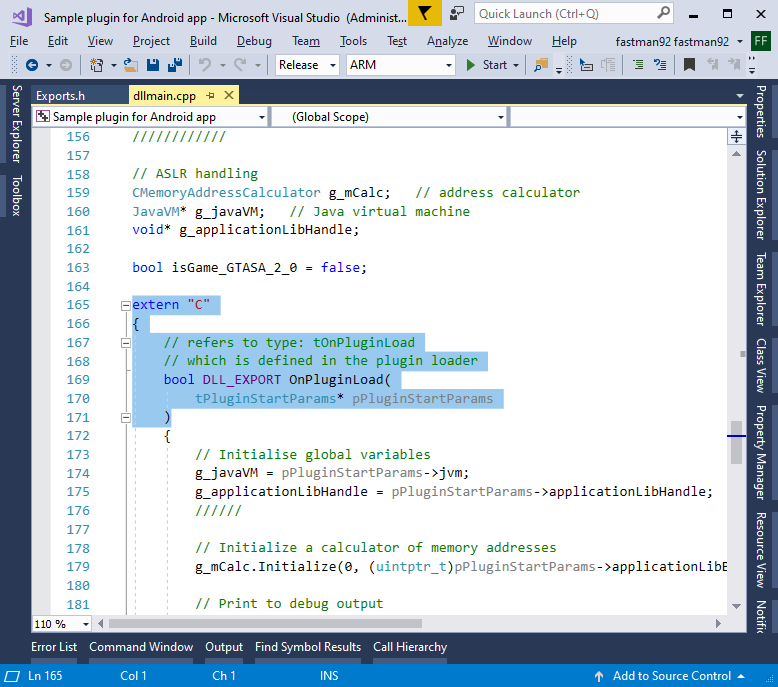
## How plugin code is executed?

It’s important to know the plugin loader executes the functions from your own plugin.

The plugin loader executes the code from plugins in the following stages:

1. Plugin loading
   * That’s a point when the application’s library has just got loaded into process, but none of the application’s library code got executed.
   * The function in the plugin to handle this stage must be named OnPluginLoad.
   * The details such as pointer to Java VM machine, plugin handle, plugin library path, application library identifier, application library base address (image base) and application library handle can be accessed and saved at this point.
   * Application library identifier (for example “SA20” – GTA SA 2.0) can be used to handle different application versions.
2. Application initialization
   * Some code of the application’s library got executed at this point.
   * The paths are set up and can be accessed by calling the functions from the plugin loader (Exports.h). The package name can also be known during this stage.

Let’s take a look, how it looks in the code of our sample plugin:

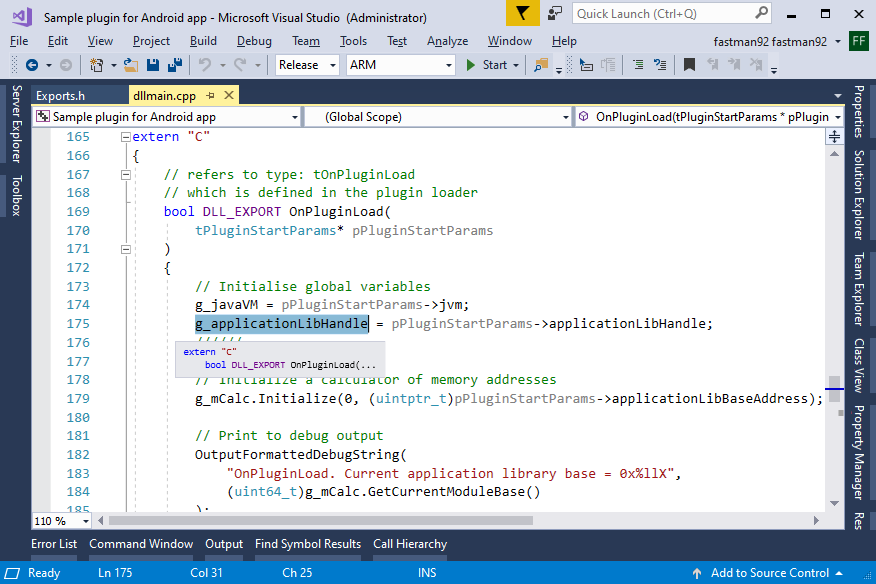


The exported functions, which should be accessed by the plugin loader need to have “C” linkage, they should have name without mangling being applied.

### OnPluginLoad

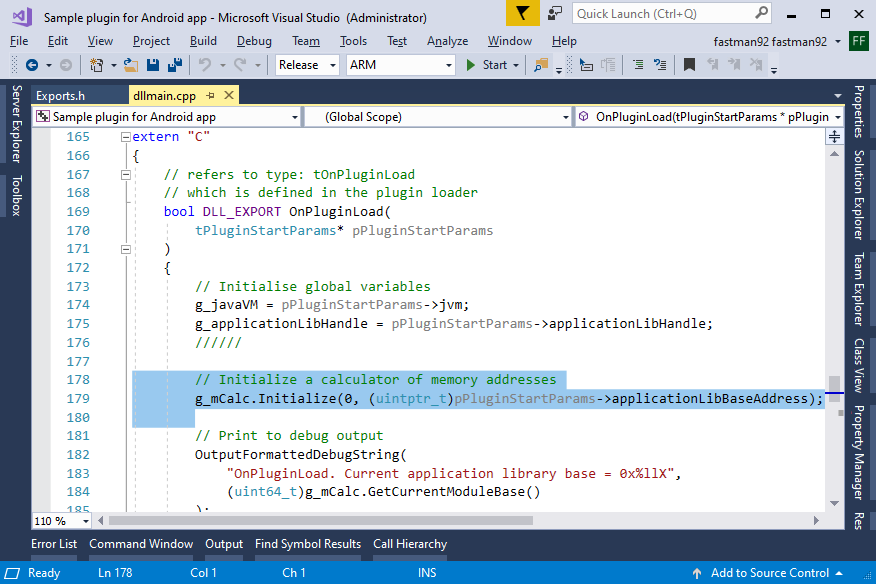
**OnPluginLoad** function will be executed by the plugin loader.

Important variables are saved to the global space by sample plugin:



g\_applicationLibHandle f, is assigned a value, which is a handle of application library (for example handle of libGTASA.so), it can be used later to get addresses of functions in the library (an argument to the **dlsym** function).

Now initialization of memory address calculator:

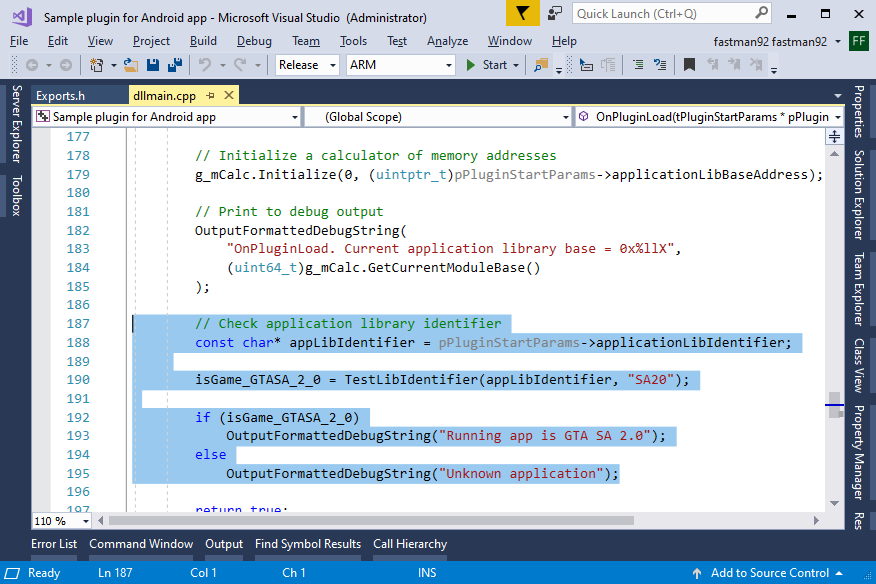


It’s important to remember about the ASLR problem, which causes the library to be loaded at random memory addresses.

The memory address calculator is initialized with the following arguments:

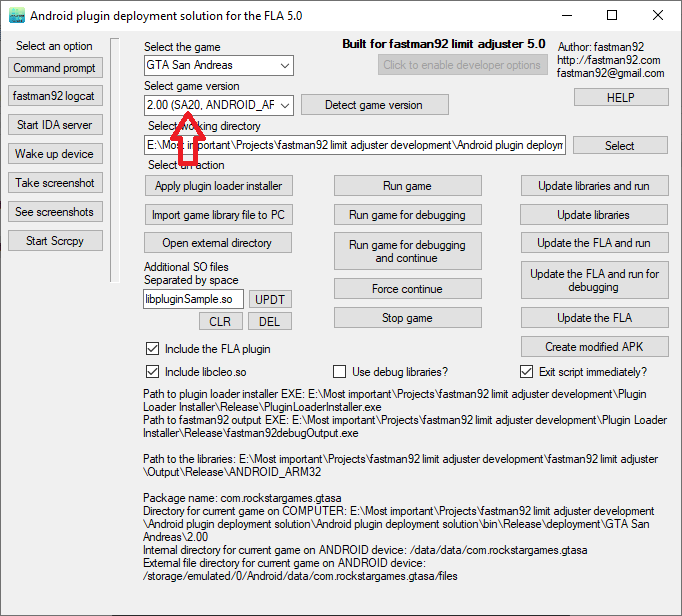
* Preferred base memory address – 0
* Current base memory address - (uintptr\_t)pPluginStartParams->applicationLibBaseAddress

The ELF libraries have a preferred base memory address 0.

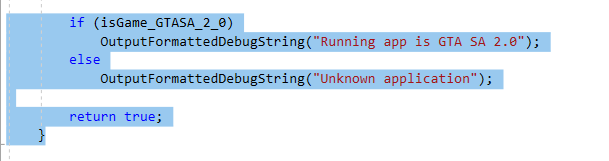


Then the application identifier is checked to determine if game version under which the plugin was started is GTA SA 2.0 – **SA20**.

The identifier can be seen in the ‘Android plugin deployment solution’ application:

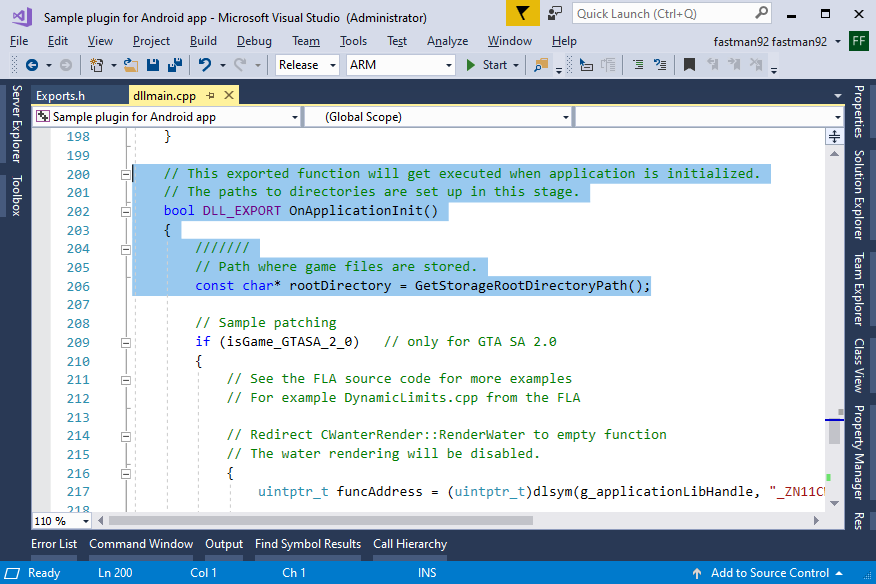


Lastly the **OnPluginLoad** function must return **true** on success or **false** on failure.



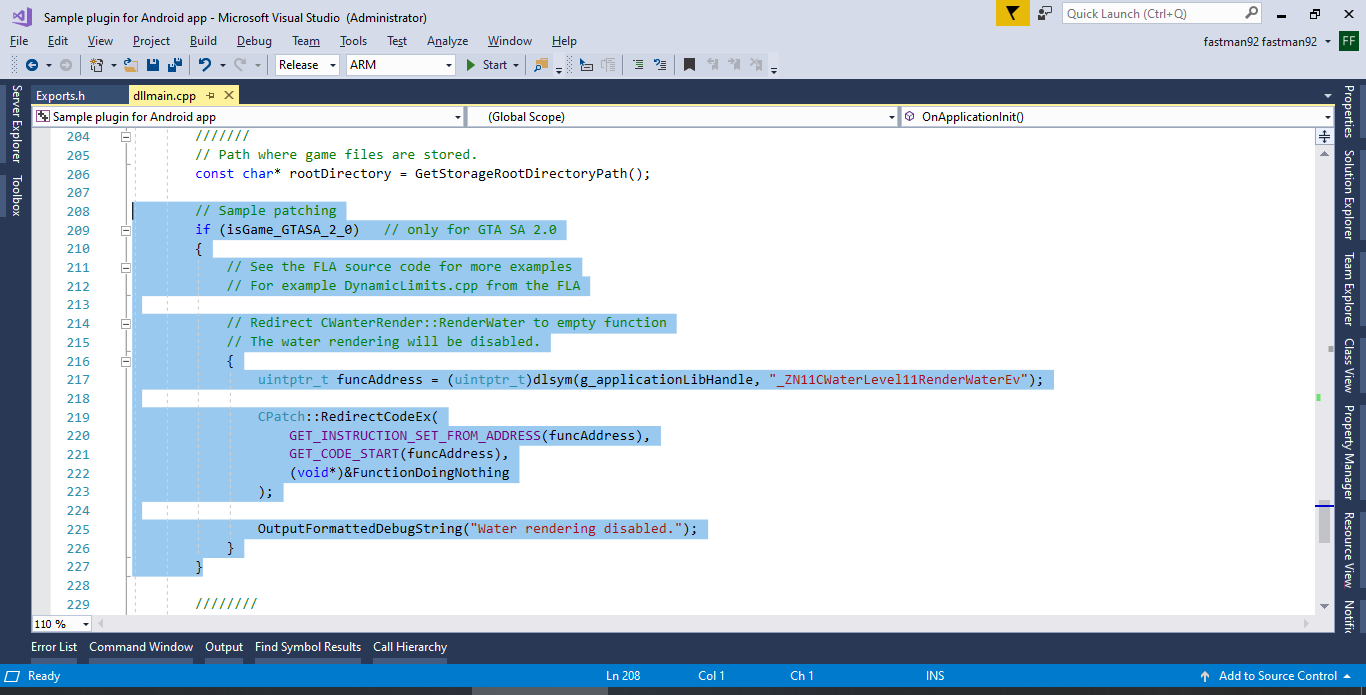
### OnApplicationInit

Function **OnApplicationInit** will be executed during the stage of application initialization.



The paths are set up during this stage, so the path to root directory can be retrieved.

Let’s do some patching for GTA SA 2.0:



CPatch::RedirectCodeEx will redirect the code from source location to target location.

[You can read the introduction on how many bytes are necessary to make a code trampoline again.](#_Introduction)

Then

uintptr\_t funcAddress = (uintptr\_t)dlsym(g\_applicationLibHandle, "\_ZN11CWaterLevel11RenderWaterEv");

Will get the jump address of CWaterLevel::RenderWater.

A jump address is an address, where the lowest bit specifying the encoding (ARM / Thumb).

[You can read the introduction on the jump addresses again.](#_Introduction)

GET\_INSTRUCTION\_SET\_FROM\_ADDRESS(funcAddress)

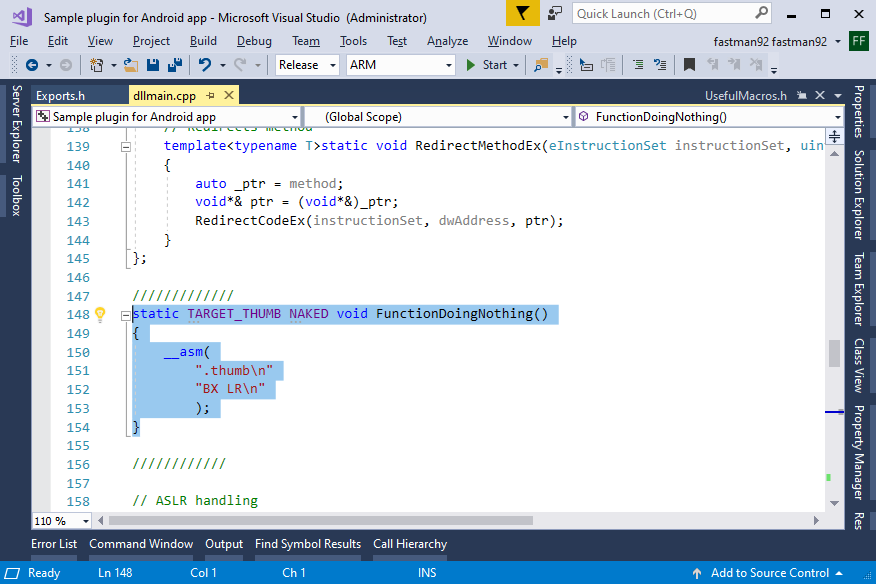
Will deduce the encoding type (ARM or Thumb) from the jump address.

GET\_CODE\_START(funcAddress)

Will get the address to the instruction from the jump address. For example if jump address points to 0x45635 (Thumb), then **GET\_CODE\_START** will return 0x45634, which is a beginning of an instruction.

(void\*)&FunctionDoingNothing

An address, to which the source location should be redirect, here an address of function, which does nothing useful:



TARGET\_THUMB fspecifier will mark the function jump address to be pointing into Thumb function.

NAKEDff specifies there should be no code in the function other than the one added in the source file. There will be no prolog or epilog added by the compiler. The function, when compiled will only have the ASM instructions, which are written in the source file and it is ("BX LR\n", nothing more.

The function has got a Thumb code, as specified later by

".thumb\n"

The code below ("BX LR\n") is compiled in Thumb mode and takes 2 bytes.

## Debugging

Debugging of game can be made possible, if you get a software ‘**IDA Pro**’ made by the company **Hex-Rays**.

The official product’s website from which it can be obtained: <https://www.hex-rays.com/>

## How to start the IDA Pro server?

The application ‘Android plugin deployment solution’ allows to start IDA Pro debugger server.

The server must have a following path on the remote Android device: **./data/bin/android\_server**

Remember, the file must have executive permissions to run it.

The debugging server can be started by **Start IDA server** button.

## How to debug the game?

Debugging the process might require an application to be started in the debug mode.

The application, which is started without the debug mode can produce an exception “Application not responding”, when touching the screen and stopped on the breakpoint.

The ANR will be disabled if an application is started in debug mode.

Here’s the correct set of steps:

1. Use ‘Android plugin deployment solution’.
2. Click button “**Run game for debugging and continue**”. The application should start on the Android device and will be waiting for the debugger.
3. Attach the debugger to the process in IDA Pro (might have to set the appropriate IP in debug settings before).

## How to debug the game, when process is starting?

When developing the plugin you may have a situation, when you want to debug how the stages **OnPluginLoad** or **OnApplicationInit**, which are executed early, when the process is starting.

Here’s the correct step of steps:

1. In IDA Pro, set the breakpoints in the interesting place, if it’s your plugin, then you might want to set a breakpoint in **OnPluginLoad** or **OnApplicationInit**
2. Use ‘Android plugin deployment solution’.
3. Click button “**Run game for debugging**”. The application should start on the Android device and will be waiting for the debugger.
4. Attach the debugger to the process in IDA Pro (might have to set the appropriate IP in debug settings before).
5. Click ‘**Force continue**’ in ‘Android plugin deployment solution’ application. The application will resume and breakpoint may be hit, if set appropriately.

# Features offered by the FLA

If you want to know if certain feature from the INI file is supported by the FLA you can open the file *Documentation.xlsx* in archive with the FLA.

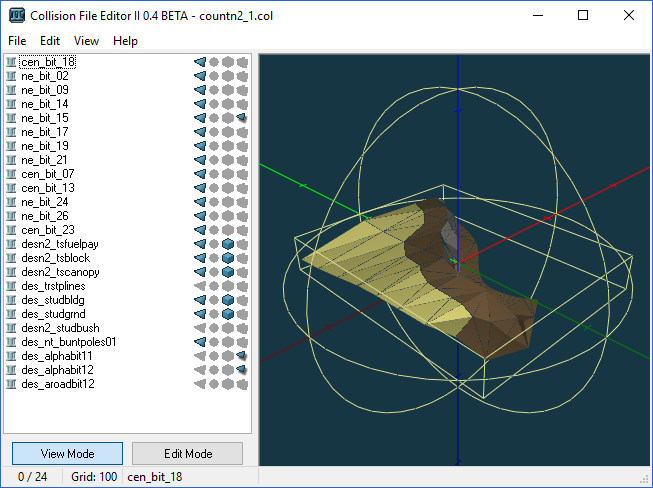
You can compare the support of options between different game versions.

## DYNAMIC LIMITS

### ColModels

The game has got COL archives.

The COL archive contains several collisions, where each is identified by different model name.

When game decides to load to load the COL archive, the game will load collisions for each of the models as identified by the collision names in the COL archive.

The screenshot shows a COL archive with 24 collisions available.

The game will crash, if this limit is exceeded.

### VehicleStructs

The memory on PS2 wasn’t so big (only 32 MB), so that Rockstar Games has decided to save the memory using an assumption that while there can exist plenty of vehicles defined, only certain amount of different vehicle models can be loaded at the time.

void CVehicleModelInfo::SetClump(RpClump \*pClump)

{

this->m\_pVehicleStruct = new CVehicleStruct;

this->CClumpModelInfo::SetClump(pClump);

/\* more code \*/

}

### rwObjectInstances

Game uses a doubly linked list to register the entities that may have to be rendered.  
If limit is exceeded, the game will not add any more objects to the list and as a consequence some of the parts of view may flicker.

### Matrices

The game allocates a matrix for the placeable objects, which can be moved as far as I have understood.  
If the limit is exceeded, the game will crash.

### PtrNode Singles

Pool of CPtrNodeSingleLink objects.

### PtrNode Doubles

Pool of CPtrNodeDoubleLink objects.

### PtrNodes

### EntryInfoNodes

### Peds

Max number of peds that can be spawned in game.

### Vehicles

Max number of vehicles that can be spawned in game.

### Objects

### Tasks

### CDummyTask

### Events

### PointRoute

### PatrolRoute

### InteriorInst

### PortalInst

### CAnimBlender

### CAtdNodeAnimChangePooledObject

### CAtdNodeAnimPlayer

### crFrameFilterBoneMask

### crFrameFilterBoneAnalogue

### crExpressionProcessor

### crmtObserver

### NavMeshRoute

### Cam

### ExplosionType

### NodeRoute

### TaskAllocator

### PedIntelligence

### PedAttractors

### QuadTreeNodes

### Collision links

### CustomEnvMapPipeMatDataPool

### CustomEnvMapPipeAtmDataPool

### CustomSpecMapPipeMaterialDataPool

## IPL

### Buildings

Max number of buildings to be loaded at any time.

The **inst** section from the IPL files creates the buildings.

There exist two types of IPL files:

* permanent IPL files
* streamed IPL files (from the IMG archives, from which the buildings may be loaded and unloaded)

### Dummies

Max number of dummy objects on the map. The buildings may be converted to dummies, when far away on the map.

### Inst entries per file

Max number of entries in the **inst** section of single IPL file.

### Entity index array

Max number of permanent IPL files.

These are text files loaded from the directory on games such as GTA III, GTA VC, GTA SA and GTA IV.

### Map zones

Max number of map zones, which are used to define levels in the game (for example Los Santos, San Fierro and Las Vegas in GTA San Andreas).

Section **zone** in the IPL file.

### Navigation zones

Max number of navigation zones, these control the text of the zone that shows up in the game as you enter a new zone.

### COccluder, apply coordinate limit patch

By default, the coordinates of occluders are limited. coordinates of occluders a

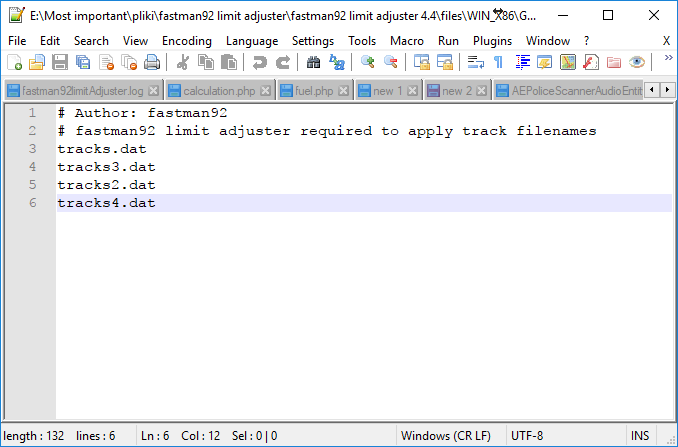
## MAP LIMITS

### Enable track config loader

This option enables the track config loader.

For GTA San Andreas the path of config file is as follows:

"data\\Paths\\gtasa\_tracks\_config.dat"



The track IDs start from 0. This means a file “tracks.dat” on the screenshot would get the ID 1, whereas the file “tracks3.dat” would get ID 2 and the file “tracks2.dat” would get the ID of 2.

You can add as many tracks as you want.

Note that a function CTrain::DoTrainGenerationAndRemoval from GTA San Andreas executable: will use the following IDs:

* Track ID 0 – for train traffic controlled by game
* Track ID 1 - for tram traffic controlled by game

Here comes a question, how to add a new track and make it usable in game, in such a way that trains or trams will be spawned on the new track.

There are two possibilities for this:

* Only for expert users: alter the function CTrain::DoTrainGenerationAndRemoval appropriately. This is going to be difficult.
* Make a script (LUA/SCM) to spawn trains on the coordinates of the newly added track.

The following command creates a train:

06D8: create\_mission\_train 13 at 2285.152 -1257.5 23.0 direction 1 store\_to 36@

You can find more train related SCM commands on this webpage:

<https://gtagmodding.com/opcode-database/browse/Train/>

In case you have problems with writing a script, you can make a question on <http://gtaforums.com>

### Enable path debugging

Enables path debugging – path nodes will be rendered in game.

To work properly, a limit of Coronas to be increased. When this option is enabled, a lot of coronas will be added to be rendered and their limit should be high enough.



### Paths map size

Paths map size can be increased by the FLA. A new set of path files will be required.

The IDs will be reallocated. See the files in **paths** directory of the FLA’s documentation.

### Radar map size

Radar map size can be increased by the FLA. A new set of radar files will be required.

The IDs will be reallocated. See the files in **radar** directory of the FLA’s documentation.

### Enable frontend map different

If enabled, two different sets of radar files will be used, like in GTA IV.

fradar\*.txd files will be drawn, when viewing a map in game menu (frontend.

radar\*.txd files will be drawn on game radar.

### World map size

Alters the world map size – the XY area of coordinates at which the statically placed objects (buildings, dummies and LOD entities) is limited.

This is because the map uses the sectors for X regions and Y regions and each region has a fixed size (as defined by World sector size).

Sample values:

* World map size = 6000

Means that the entities may be placed in coordinates, which meet this condition:

coord >= -3000.0 && coord < 3000.0;

* World map size = 12000

Means that the entities may be placed in coordinates, which meet this condition:

coord >= -6000.0 && coord < 6000.0;

## LEVEL LIMITS

These limits allow to define more levels in game.

Each level defines the following:

* Police
  + Cop car
  + Cop ped
* Ambulance
  + Ambulance car
  + Medic ped
* Fire
  + Fire truck car ID
* Taxi
  + Cab driver ped ID
* Item price multiplier

## VEHICLE SPECIAL FEATURES

This is an old feature, made to enable special features for vehicle models, but limit to ZR350 and HYDRA abilities. Retained for backward compatibility with old INI files.

Use a loader of [model special features](#_Model_special_features), which can apply special features of any model ID.

## IMG LIMITS

### Enable handling of new enhanced IMG archives

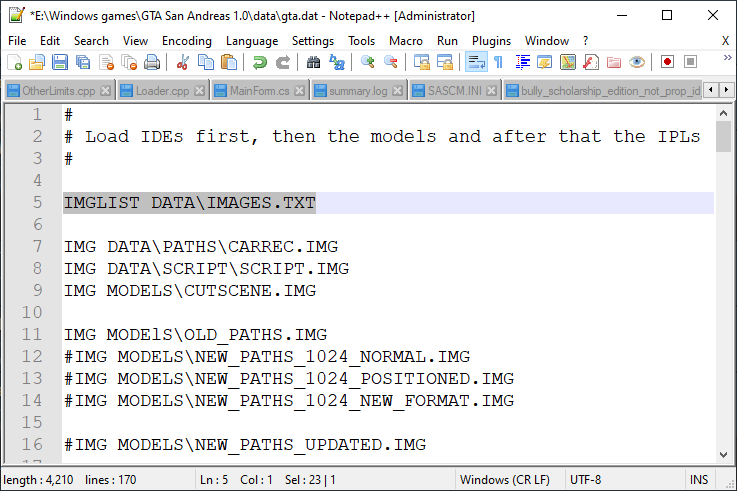
Enables handling of enhanced IMG archives.

You can read more about them [here](#_Enhanced_IMG_format).

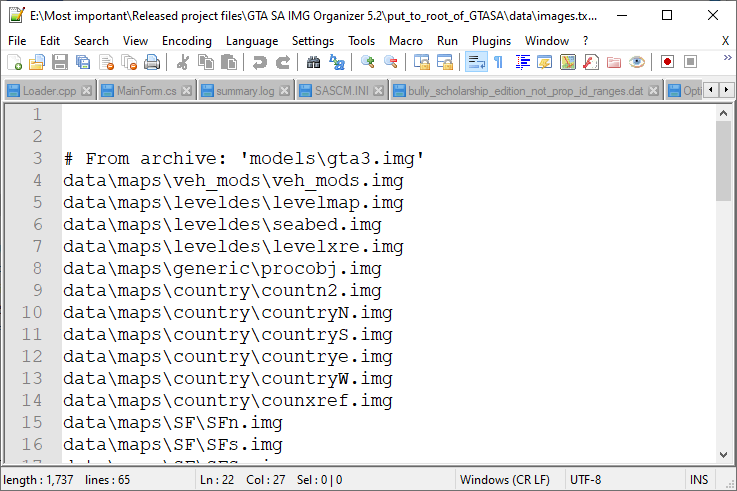
### Enable handling of IMGLIST keyword

Adds a possibility to use IMGLIST keyword in level file such as gta.dat or default.dat

Example:



images.txt:



## ID LIMITS

IMG archives from game contain files of few different types, distinguished by their extension.

Each file will need its own ID, the number of IDs.

**GTA SA 1.0 US HOODLUM**

These limits are best developed for GTA San Andreas 1.0 US HOODLUM, as the log might say:

* Count of file IDs is over 32767 and requires unsigned ID. Applying unsigned ID patches.
* Count of file IDs is over 65532 and base ID of CStreamingInfo requires int32\_t. Applying int32\_t base ID patches.
* Count of DFF/TXD IDs is over 65532 and requires int32\_t. Applying int32\_t DFF+TXD ID patches.
* COL ID limit is over 256 and requires something more than uint8\_t. Applying COL ID uint32\_t patches.
* IPL ID limit is over 256 and requires something more than uint8\_t. Applying IPL ID uint32\_t patches.

New ID limits:

0 - 59999 (60000) - DFF models defined within IDE files

60000 - 119999 (60000) - TXD texture archives.

120000 - 120554 (555) - COL collision archives.

120555 - 121110 (556) - IPL Binary IPL files.

121111 - 121174 (64) - DAT files limited to nodes\*.dat

121175 - 121654 (480) - IFP animation archives.

121655 - 122229 (575) - RRR car recordings, carrec\*.rrr files

122230 - 122329 (100) - SCM scripts

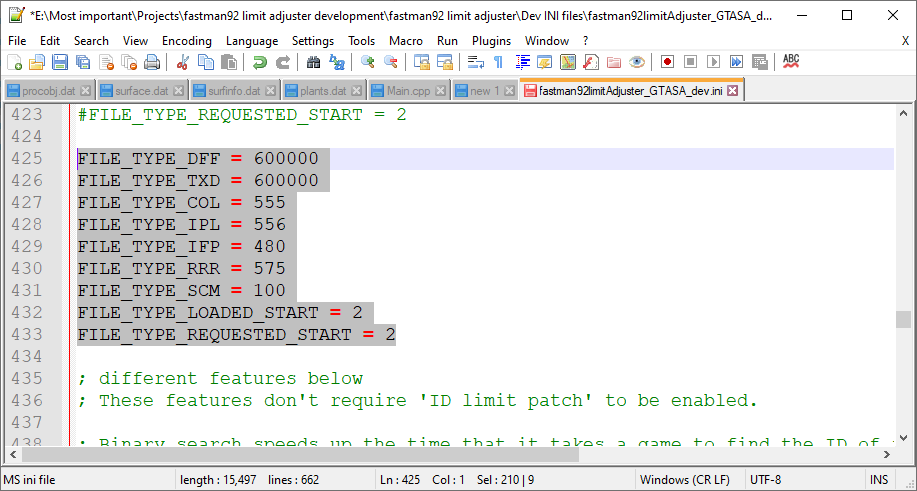
122330 - 122331 (2) - Loaded list

122332 - 122333 (2) - Requested list

122334 - count of all file IDs

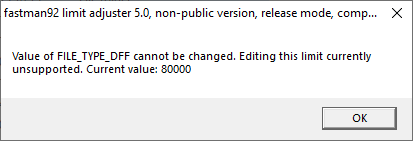
The number of IDs in GTA San Andreas 1.0 HOODLUM is only limited by the amount of memory.

Example of very high limits:



Game contains IDE files or IDB files (Bully Scholarship Edition).

If the FLA is unable to change certain limit to the specified, it will show a following error on Windows:



Similarly, an error will be produced, if the value of ID limit is set too high.

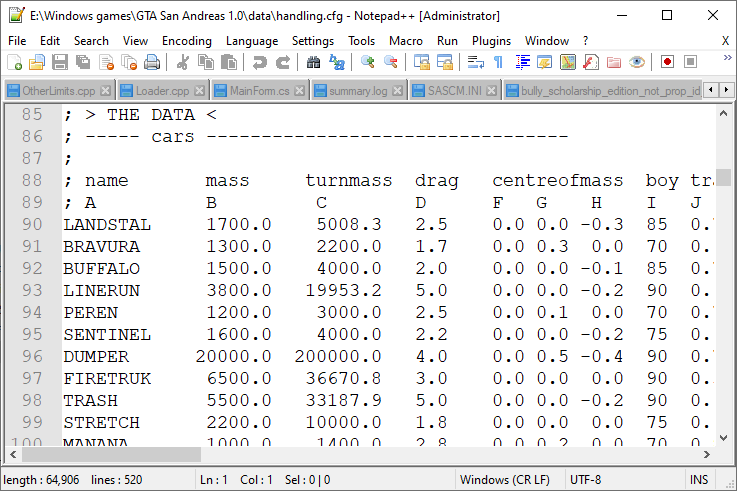
### Apply ID limit patch

This setting enables processing of ID limits.

Must be enabled, if ID limits need to be changed.

## HANDLING.CFG LIMITS

File:



Game have limits on the amount of lines of various types in handling.cfg / handling.dat.

*Moreover, games such as Bully SE, GTA III, GTA VC and GTA SA have a fixed list of identifiers.*

*Adding any new identifier will result in bugs, different identifier will not work.*

*This is not the problem if handling.cfg limits are enabled. The FLA can handle any identifier.*

Here’s an explanation of handling lines:

* standard lines – that’s the most basic data and required for every identifier
* bike lines – used for vehicles, which can act as bikes
* flying lines – used for vehicles, which can fly (planes or helicopters come to mind)
* boat lines – used for vehicles, which can move in water

## OTHER LIMITS

### LOD distance

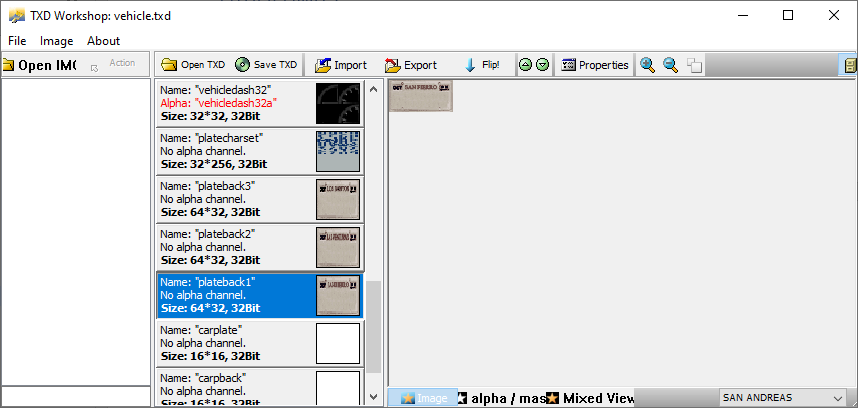
The maximum object distance to which the object can have a collision.

### Number of plate textures

Exact number of plateback textures in **vehicle.txd** file.

If you put 3 plateback textures in vehicle.txt, then set this value to 3, if you put 5 plateback textures, then set the value to 5.

Path to file in GTA San Andreas: models\generic\vehicle.txd

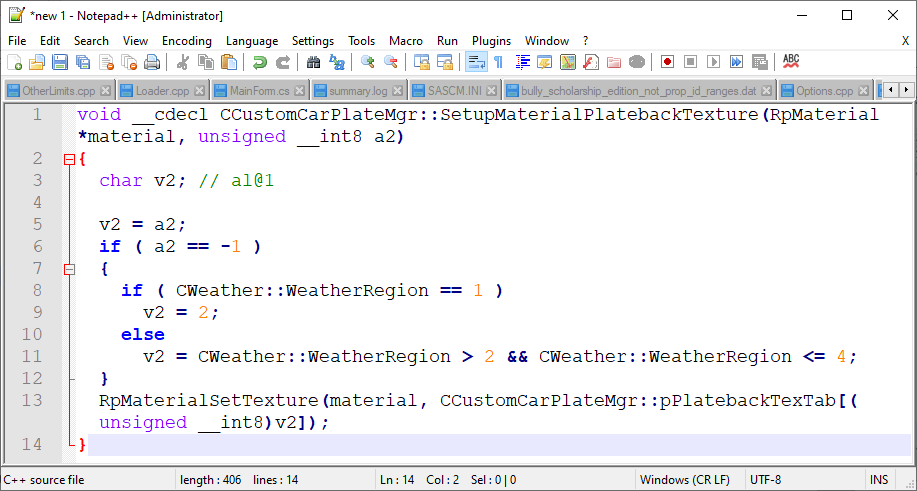


The textures must exist.

You might also want to write a plugin, which modifies a function responsible for default assignment of ‘plateback’ texture depending on weather region: CCustomCarPlateMgr::SetupMaterialPlatebackTexture

An address to this function in GTA SA 1.0 US: **0x6FDE50**

Default code of this function:



### Coronas

Changes the max number of coronas that may rendered per frame.

## ROADBLOCK LIMITS

## Apply roadblox.dat better loader

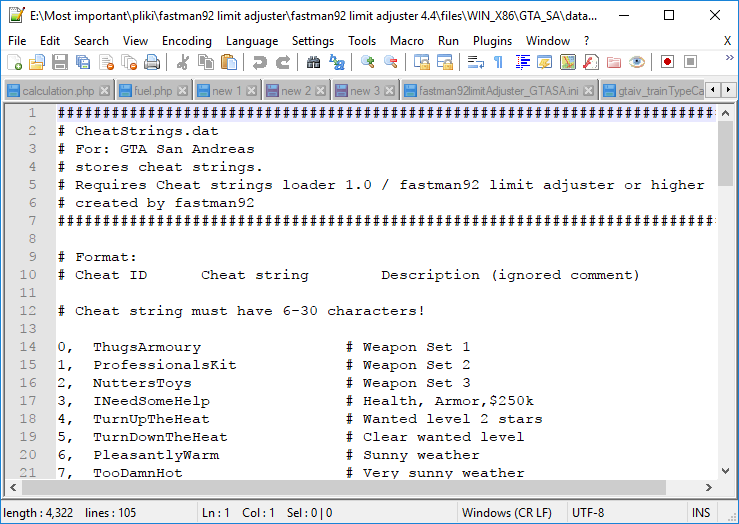
This option to use a roadblox.dat of variable size.

## ADDONS

### Cheat String Loader

Cheat string loader allows to alter the cheat strings, as the name of the feature should imply.

The file format is very simple and doesn’t require an explanation:



### Vehicle audio loader

Vehicle audio loader allows to set the audio settings for specified vehicle model names.

#### GTA Vice City / GTA San Andreas

The single .cfg file is used to configure the vehicle settings.

The settings for one vehicle are specified in one line.

#### GTA IV / GTA EFLC

A big .ini file with many options is used to configure the vehicle settings.

The file that always gets loaded comes under the path "pc\\audio\\Config\\gtaiv\_vehicleAudioSettings.ini".

The vehicle audio loader may load TLAD or TBoGT settings additionally, when user chooses one of these.

If you’re using the GTA EFLC EXE, the paths are like this:

* "DLC\\TBoGT\\content\\pc\\audio\\config\\TBoGT\_vehicleAudioSettings.ini"
* "DLC\\TLAD\\content\\pc\\audio\\config\\TLAD\_vehicleAudioSettings.ini"

If you’re using the GTA IV EXE, the paths are like this:

* "TLAD\\pc\\audio\\config\\TLAD\_vehicleAudioSettings.ini"
* "TBoGT\\pc\\audio\\config\\TBoGT\_vehicleAudioSettings.ini"

Basically "\\content\\" is added in the path, when using the GTA IV EXE with the DLCs.

### FXT loader

Inspired by the FXT loader that came with CLEO for GTA San Andreas the FLA has got the FXT loader implemented for GTA IV and GTA V.

The FXT replaces the text for specified GXT labels, using one text no matter what the selected language in game.

The FXT loader doesn’t support adding different texts for different languages.

If you enabled the FXT loader in the INI, you can start using it by creating the text files in the directory FXT\ in game root. The use of the .fxt extension is recommended, although not required.

The syntax of FXT files is very simple.

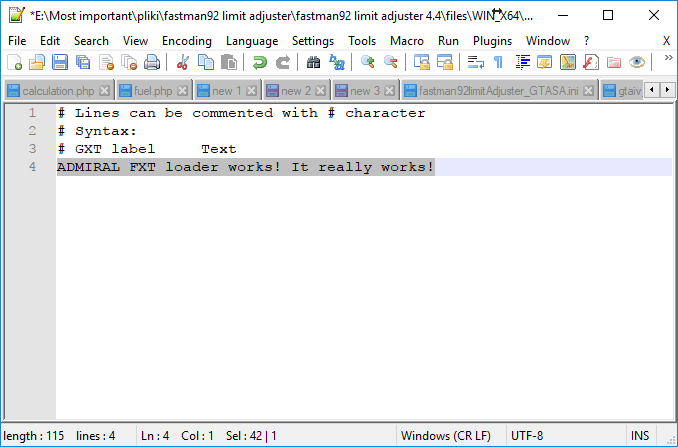
The line, which changes a text for GXT label ADMIRAL:

ADMIRAL FXT loader works! It really works!

The comment line:

# some comment

Example:

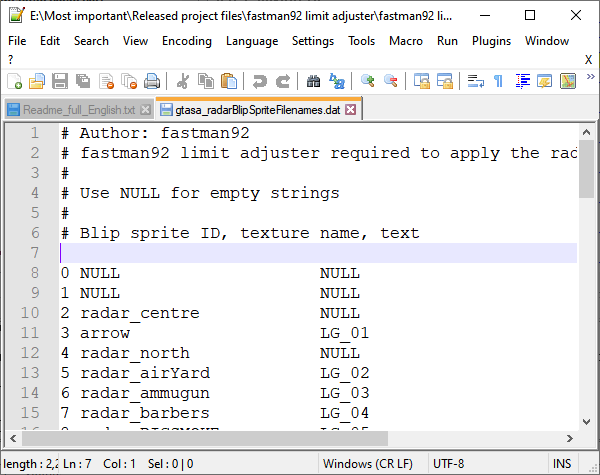


### Radar blip sprite filename loader

This loader allows to add new radar sprites to game.

These can be used with SCM commands, which create blips on map, for example:

*02A8: add\_sprite\_blip\_for\_coord $X\_JOHNSON\_HOUSE $Y\_JOHNSON\_HOUSE $Z\_JOHNSON\_HOUSE sprite $ICON\_CJ store\_to $439*  
  
**sprite** is the key argument here. It’s an sprite ID.



Every sprite has got following properties:

1. Blip sprite ID
2. Texture name (from hud.txd)
3. GXT label – used on map on the legend

For empty strings **NULL** value should be used.

The textures need to be placed inside of models\hud.txd

### Train type carriages loader

This loader allows to control the following array from an executable: TrainTypeCarriages.

It’s possible to add or remove train type carriages.

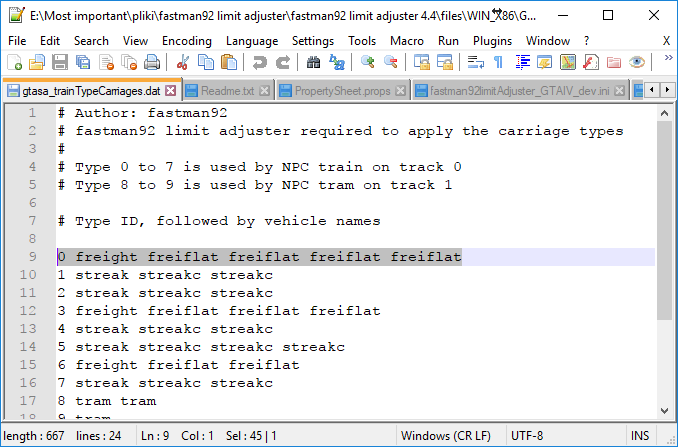
The game has defined many different train type carriages.

What does it define?

The train type carriage defines a set of trains connected one to another.

The first train controls all the rest of carriages.

Consider the following example:



The selected line defines a type with ID of 0, with the elements “freight freiflat freiflat freiflat freiflat”

The result is going to be like this:

If you want to make the new IDs usable in game, read the following: [HowToAddUsableTrains](#HowToAddUsableTrains)

### Model special features loader

Game might assign certain behavior to specified model IDs.

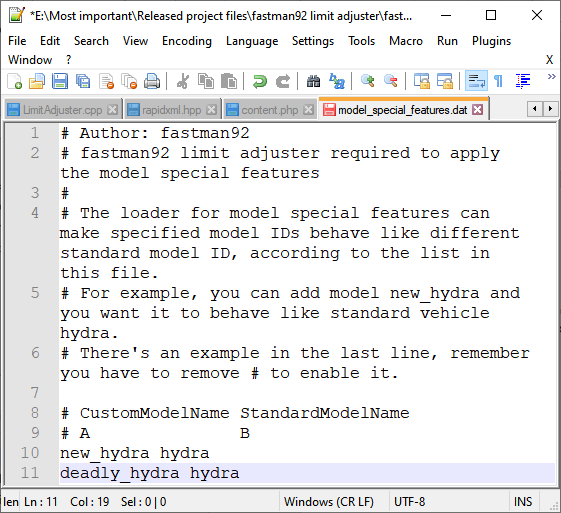
For example, hydra in GTA San Andreas has got an ID 520.

Consequently, the model ID 520 can do vertical take-off and landing (VTOL).

Only one model (520) can do VTOL by default. *Now if we added a new vehicle, which looks like a fighter jet, it still would be missing a proper behavior.*

**With a long development and constant improvements, the FLA has finally reached a point, when it became possible to assign behavior of one model ID to another.**

When the loader of model special features is enabled as defined by the INI, the FLA will read the file "data\\model\_special\_features.dat".  
Let’s take a look at simple example:



The file on the screenshot will cause the following:

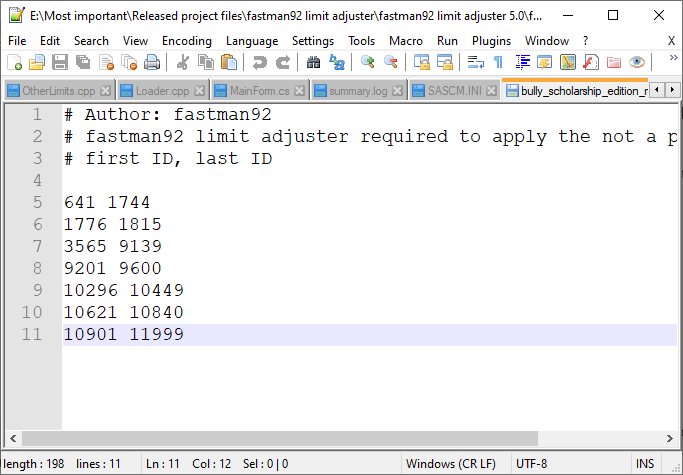
* Model named **new\_hydra** will act like model **hydra**
* Model named **deadly\_hydra** will act like model **hydra**

Though the example only shows **hydra**, you can actually put any other model name in this file.

### Not a prop ID range loader

Not a prop ID range loader for Bully SE allows to define ranges of model IDs, which will be considered unique.

These can only be spawned once in game, while the props can be spawned multiple times.



**fastman92 processor** can be used to decode/encode Bully SE’s IDB and IPB files and see more information.

## MAIN

### Use a different INI

This option allows to use a different configuration file specified by the path. Primarily used during the development.

On Windows the path supports Windows environment variables such as **%SYSTEMDRIVE%**

# CLEO for Android

fastman92 limit adjuster targets the problem of CLEO made by Alexander Blade lacking a support of newest GTA versions on Android.

The FLA archive includes the following version of libcleo.so: **CLEO ANDROID (04 May 2016)**

The FLA requires can patch the following CLEO versions:

* **CLEO ANDROID (04 May 2016)**
* **CLEO ANDROID (24 Mar 2015)**

Versions:

* 0 - undefined
* 1 - GTA3 v1.4
* 2 - GTAVC v1.03
* 3 - GTASA v1.00
* 5 - GTASA v1.02
* 6 - GTASA v1.03
* 7 - GTASA v1.05
* 8 - GTASA v1.06
* 9 - GTASA v1.05 GER
* 10 - GTASA v1.07
* 11 - GTA3 v1.6
* 12 - GTAVC v1.06
* 13 - GTASA v1.08

Versions IDs added by the FLA:

* 14 - GTA3 v1.8
* 15 - GTAVC v1.07, not implemented by the FLA
* 16 - GTAVC v1.09
* 17 – GTASA v2.00

# Enhanced IMG format

The format of enhanced IMG archives was implemented to account for the following:

* Possible compression of individual files
* Possible encryption of file list and individual files
* The original size in bytes may be calculated
* The packed size in bytes may be calculated
* Support for nested IMG archives, the child IMG file in the parent IMG archive should have no compression and no encryption, otherwise the FLA will show an error

## How to make archives in enhanced IMG format?

It’s possible to read and write these archives with an application **fastman92 processor**.

## Test of archive:

All files from gta3.img, which comes with GTA San Andreas have been compressed with specified compression type and fully rebuilt.

IMG\_FASTMAN92\_GTASA\_COMPRESSION\_TYPE\_NONE - 915 MB archive

IMG\_FASTMAN92\_GTASA\_COMPRESSION\_TYPE\_ZLIB, highest compression level - 533 MB archive

IMG\_FASTMAN92\_GTASA\_COMPRESSION\_TYPE\_LZ4, highest compression level, LZ4 HC actually - 686 MB archive

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

## Structure of IMG archive.

beginning of IMG archive, first header:

struct tImgFastman92GTASAfirstHeader

{

uint32\_t magicID; // VERF

uint32\_t archiveFlags; // bits 0-3 archive version (should be 1), bits 4-7 metadata encryption type, bits 8-10 game type

char authorName[12]; // constant value "fastman92" padded with zeros.

};

VALIDATE\_SIZE(tImgFastman92GTASAfirstHeader, 0x14);

Second header, could be encrypted:

struct tImgFastman92GTASAsecondHeader

{

uint32\_t Check; // constant value 1, after decryption it should have value 1. To check if decryption worked properly.

uint32\_t NumberOfEntries;

char reserved[8]; // for future use, should be NULL now.

};

VALIDATE\_SIZE(tImgFastman92GTASAsecondHeader, 16);

Now there comes a list of entries, the entries may be encrypted, each entry of type:

class tImgFastman92GTASAdirectoryEntry

{

public:

uint32\_t PositionInSectors;

uint16\_t OriginalSizeInSectors;

uint16\_t NumberOfPaddedBytesInAlignedOriginalSize;

uint16\_t PackedSizeInSectors;

uint16\_t NumberOfPaddedBytesInAlignedPackedSize;

uint32\_t Flags;

char Name[IMG\_FASTMAN92\_GTASA\_MAX\_FILENAME\_LENGTH + 1];

char Reserved[8]; // should have NULL values when reserved.

};

VALIDATE\_SIZE(tImgFastman92GTASAdirectoryEntry, 64);

Enumerations:

Game type (3 bits for this value)

* 0: GTA SA

Compression type (4 bits for this value):

* 0: no compression
* 1: ZLIB compression
* 2: LZ4 / LZ4 HC compression

Encryption type (4 bits for this value):

* 0: no encryption
* 1: encryption of variation 1, discouraged
* 2: encryption of variation 2

That’s all!

# Path format

GTA San Andreas

## Header

The FLA supports two types of path files:

* standard GTA SA path files
* fastman92 path files:
  + **FLA4 – good and simple format**
  + FM92 – obsolete, unneccessarily complicated format
    - VER2
    - VER3

**FLA4** is the format in which it’s recommended to create path files.

Use **FLA4** format. This is the newest and best one.

### The first part

In the beginning of file there is tPathFileHeaderFirstPart structure:

struct tPathFileHeaderFirstPart

{

union

{

uint32\_t numberOfPathNodes;

uint32\_t magicID; // magicID = 0xFFFFFFFF when different path format.

};

};

If the path file is extended, the member magicID from tPathFileHeaderFirstPart will have a value of exactly 0xFFFFFFFF. Otherwise, it’s a standard and at the beginning there’s a structure tStandardPathFileHeaderCounts – see: The numbers of elements

### Version specific header

#### In case of version " FLA4"

uint32\_t format; // should have a value "FLA4"

value '4ALF' as can be seen

#### In case of version "FM92", unnessarily complicated

struct tFastman92PathFileSecondPart

{

uint32\_t format; // should have a value "FM92"

uint8\_t sizeofNextValue;

char value[]; // number of bytes depends on sizeofNextValue

};

Some information on members:

* format should have a value '29MF'
* The allowed value: "\x00" "fastman92" "\x00"
* The size of allowed value: 11. sizeofNextValue should have a value 11

### The numbers of elements

Finally the header contains these fields:

// Elements counts in path file

struct tStandardPathFileHeaderCounts

{

uint32\_t numberOfPathNodes;

uint32\_t numberOfPathVehicleNodes;

uint32\_t numberOfPathPedNodes;

uint32\_t numberOfCarPathLinks;

uint32\_t numberOfAdjacentAddresses;

};

namespace details

{

struct tFastman92PathFileHeaderThirdFormat

{

uint32\_t formatVersion; // should have a value "VER2" or "VER3"

};

}

struct tFastman92PathFileHeaderThirdPart

: details::tFastman92PathFileHeaderThirdFormat,

tStandardPathFileHeaderCounts {};

The third part contains formatVersion, which should be “VER2” or “VER3”, preferably the newest.

tFastman92PathFileHeaderThirdPart also contains the numbers of elements, like number of path nodes, number of car path nodes and so on.

## Path nodes

The vehicle path nodes come first. The ped path nodes come later.

The number of elements is defined by numberOfPathNodes from tStandardPathFileHeaderCounts.

Their structure:

// standard GTA SA

class CPathNode

{

public:

CPathNode \*m\_pPrev;

CPathNode \*\*m\_ppNext;

CompressedVector m\_posn; // unused when fastman92 path format VER2 or higher is used

int16\_t m\_wSearchList;

int16\_t m\_wConnectedNodesStartId; // ID into link array

CNodeAddress m\_nodeInfo;

char m\_nPathWidth;

uint8\_t m\_nFloodID; // unused when fastman92 path format VER3 or higher is used

uint32\_t m\_dwFlags;

// Returns number of links

int GetNumberOfLinks() { return this -> m\_dwFlags & 0xF; }

// Sets the position in vector

void GetPosition(CVector& out)

{

out.x = (float)this->m\_posn.x / PATH\_COORD\_MULTIPLIER;

out.y = (float)this->m\_posn.y / PATH\_COORD\_MULTIPLIER;

out.z = (float)this->m\_posn.z / PATH\_COORD\_MULTIPLIER;

}

};

VALIDATE\_SIZE\_ONLY\_ON\_32\_BIT\_ARCHITECTURE(CPathNode, 0x1C);

// fastman92 path format VER2

class CPathNode\_fastman92\_version\_2 : public CPathNode

{

public:

CompressedVector\_extended m\_extended\_posn;

// Sets the position in vector

void GetPosition(CVector& out)

{

out.x = (float)this->m\_extended\_posn.x / PATH\_COORD\_MULTIPLIER;

out.y = (float)this->m\_extended\_posn.y / PATH\_COORD\_MULTIPLIER;

out.z = (float)this -> m\_extended\_posn.z / PATH\_COORD\_MULTIPLIER;

}

};

VALIDATE\_SIZE\_ONLY\_ON\_32\_BIT\_ARCHITECTURE(CPathNode\_fastman92\_version\_2, 0x28);

// fastman92 path format VER4

class CPathNode\_fastman92\_version\_4 : public CPathNode\_fastman92\_version\_2

{

public:

uint16\_t m\_nExFloodID;

};

VALIDATE\_SIZE\_ONLY\_ON\_32\_BIT\_ARCHITECTURE(CPathNode\_fastman92\_version\_4, 0x2C);

// fastman92 path format VER2

class CPathNode\_fastman92\_version\_2 : public CPathNode

{

public:

CompressedVector\_extended m\_extended\_posn;

// Sets the position in vector

void GetPosition(CVector& out)

{

out.x = (float)this->m\_extended\_posn.x / PATH\_COORD\_MULTIPLIER;

out.y = (float)this->m\_extended\_posn.y / PATH\_COORD\_MULTIPLIER;

out.z = (float)this -> m\_extended\_posn.z / PATH\_COORD\_MULTIPLIER;

}

};

VALIDATE\_SIZE\_ONLY\_ON\_32\_BIT\_ARCHITECTURE(CPathNode\_fastman92\_version\_2, 0x28);

// fastman92 path format VER4

class CPathNode\_fastman92\_version\_4 : public CPathNode\_fastman92\_version\_2

{

public:

uint16\_t m\_nExFloodID;

};

VALIDATE\_SIZE\_ONLY\_ON\_32\_BIT\_ARCHITECTURE(CPathNode\_fastman92\_version\_4, 0x2C);

// fastman92 path format VER3

// comes with alignment problem, version 4 fixes it

#pragma pack(push, 1)

class CPathNode\_fastman92\_version\_3 : public CPathNode\_fastman92\_version\_2

{

public:

uint16\_t m\_nExFloodID;

};

#pragma pack(pop)

VALIDATE\_SIZE\_ONLY\_ON\_32\_BIT\_ARCHITECTURE(CPathNode\_fastman92\_version\_3, 0x2A);

// comes with alignment problem, version 4 fixes it

#pragma pack(push, 1)

class CCarPathLink\_fastman92\_version\_2 : public CCarPathLink

{

public:

struct {

int32\_t x;

int32\_t y;

} extended\_pos;

};

#pragma pack(pop)

VALIDATE\_SIZE(CCarPathLink\_fastman92\_version\_2, 0x16);

Some explanation:

* CPathNode is used in standard path file
* CPathNode\_fastman92\_version\_2 is used in fastman92 path file with FM92:VER2 version, the limited coordinate fields became depreciated and new 32-bit coordinate fields got added – m\_extended\_pos member.
* CPathNode\_fastman92\_version\_3 is used in fastman92 path file with FM92:VER3 version. The 8-bit flood fill ID got depreciated and 16-bit field named m\_nExFloodID got added.
* CPathNode\_fastman92\_version\_4 is used in fastman92 path file with FLA4 version. Alignment problem fixed. **You should prefer this one.**

## Car path links

The car path links are used for the vehicle nodes. They were introduced to make a vehicle traffic going smoother, using Bezier lines with 3 points.

The number of elements is defined by numberOfCarPathLinks from tStandardPathFileHeaderCounts.

Structure used for each element:

class CCarPathLink

{

public:

struct

{

int16\_t x;

int16\_t y;

} pos; // unused when path limit hacked

CNodeAddress attachedPathNode;

int8\_t dirX;

int8\_t dirY;

char m\_nPathNodeWidth;

uint16\_t m\_wFlags;

};

VALIDATE\_SIZE(CCarPathLink, 0xE);

class CCarPathLink\_fastman92\_version\_4 : public CCarPathLink

{

public:

struct {

int32\_t x;

int32\_t y;

} extended\_pos;

};

VALIDATE\_SIZE(CCarPathLink\_fastman92\_version\_4, 0x18);

// comes with alignment problem, version 4 fixes it

#pragma pack(push, 1)

class CCarPathLink\_fastman92\_version\_2 : public CCarPathLink

{

public:

struct {

int32\_t x;

int32\_t y;

} extended\_pos;

};

#pragma pack(pop)

VALIDATE\_SIZE(CCarPathLink\_fastman92\_version\_2, 0x16);

* CCarPathLink is used in standard path file
* CCarPathLink\_fastman92\_version\_2 is used in fastman92 path file with FM92:VER2 / FM92:VER3 version, the limited coordinate fields became depreciated and new 32-bit coordinate fields got added - extended\_pos member.
* CCarPathLink\_fastman92\_version\_4 is used in fastman92 path file with FLA4 version. Alignment problem fixed. **You should prefer this one.**

## Adjacent path node addresses

These are addresses to adjacent nodes, **4 bytes** per entry.

The number of elements: tStandardPathFileHeaderCounts.numberOfAdjacentAddresses + 192.

Structure used for each element:

struct CNodeAddress

{

uint16\_t areaId;

uint16\_t nodeId;

};

## Adjacent car path link addresses

These are links to adjacent car path links. These are only set for the vehicle paths.

The number of elements is defined by numberOfAdjacentAddresses from tStandardPathFileHeaderCounts.

Standard path format uses this structure:

class CLinkAddress

{

public:

unsigned short m\_wCarPathLinkId : 10;

unsigned short m\_wAreaId : 6;

};

VALIDATE\_SIZE(CLinkAddress, 0x2);

fastman92 path format uses this structure:

class CLinkAddress\_extended

{

public:

unsigned short m\_wCarPathLinkId;

unsigned short m\_wAreaId;

};

VALIDATE\_SIZE(CLinkAddress\_extended, 0x4);

## Distance to adjacent path node addresses

These are the distances between linked nodes in full units, 1 byte per entry. They are essential for path finding algorithms.

The number of elements: tStandardPathFileHeaderCounts.numberOfAdjacentAddresses + 192.

Type of each element: uint8\_t

## Path Intersection Flags

This section consists of intersection flag values for each node address (i.e. node link).

The number of elements: tStandardPathFileHeaderCounts.numberOfAdjacentAddresses + 192.

Structure used for each element:

class CPathIntersectionInfo

{

public:

unsigned char m\_bRoadCross : 1;

unsigned char m\_bPedTrafficLight : 1;

};

VALIDATE\_SIZE(CPathIntersectionInfo, 0x1);

## EOF

Only applicable to fastman92 path file:

File should be terminated by an uint32\_t value 'FOE'. Otherwise, it's considered invalid.

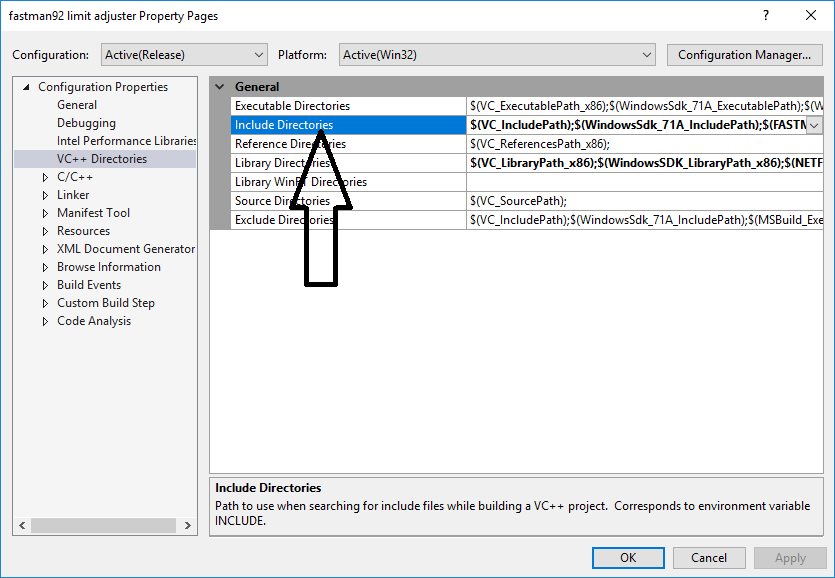
# How to make projects compatible with the FLA?

## Integrate ForOtherProjects into your project

**ForOtherProjects** is set of files, which allow you to call the FLA functions. There exists a set of files, which is made to work with/without the FLA as well.

Find the directory in RAR: source code\fastman92 limit adjuster\fastman92 limit adjuster\Source files\ForOtherProjects

Add this directory to your project – you can copy the files or add it to Include directories:



**ForOtherProjects** requires three files from **fastman92 include**, because of an include in *Main.h*:

#include <Library\Library.h>

You need to get these files for your project:

fastman92 include\PlatformGeneralisation.h

fastman92 include\Library\Library.h – add it to list of your header files if you want

fastman92 include\Library\Library.cpp – you must add it to list of your source files

Because my files are designed to work on multiple platforms, you have set up certain preprocessor macros in your project, which tell something about the platform.

See “fastman92 include\Info.txt” and SET the necessary preprocessor macros in your project.

### Use FLA\_not\_required

The **ForOtherProjects** contains a directory **FLA\_not\_required**, which includes the files made in such a way that they work with/without the FLA. This is highly recommended to use.

Add the files from **ForOtherProjects\FLA\_not\_required** to your project – header and source files.

You need to initialize the features from **FLA\_not\_required**, before they can be used in your project.

In the project, you must add #include <FLA\_not\_required/Main.h>

Lastly you should execute a one function to perform initialization:

CFastman92limitAdjuster::Init();

After you do this, the features from **FLA\_not\_required** are ready to be used.

### Use FLA\_required

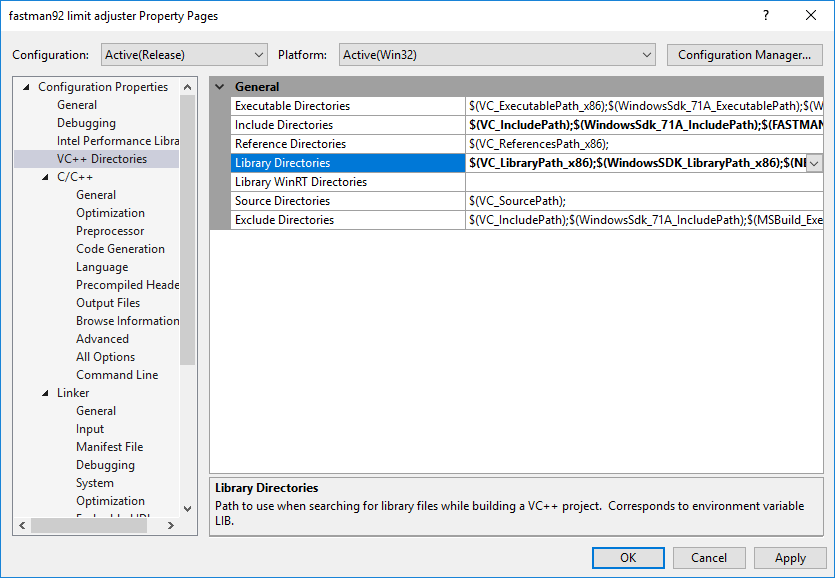
The **ForOtherProjects** contains a directory **FLA\_required**, which contains declarations of the exported functions, which are a part of the FLA. If you use these functions, your project will be importing the functions from the FLA library and the FLA will be required as a result. This is not recommended to use.

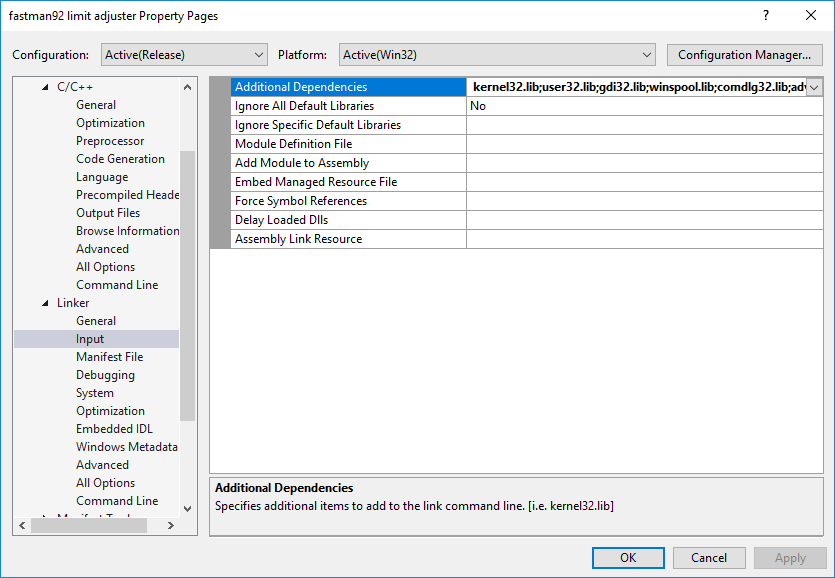
If you still want to use **FLA\_required**, you should include this file whenever you want to use exported functions:

#include <FLA\_required/Exports.h>

The compiler will also require a **lib** file if you have used any of the FLA exported functions.

Related settings:





Where can you find the FLA’s **lib** file?

For example, files\WIN\_X86\plugin\$fastman92limitAdjuster.lib

For WIN\_X64 the path will be a little different, use an analogy.

## ID limits

### Table of possible IDs

This table relates to the problem of model/texture archive/file IDs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **Normal IDs** | **Undefined ID** | **Unknown -2 ID** | **Info** |
| int16\_t | lowest 0, highest 32767 | -1 | -2 | Default in game. |
| uint16\_t | lowest 0, highest 0xFFFC | 0xFFFF | 0xFFFE | Brought when the FLA became more advanced. |
| int32\_t | lowest 0, highest 0x7FFFFFFF | -1 | -2 | The final step of evolution of the FLA on the IDs. Use CFileID in the game structures. This is the best option. |

Rockstar Games made the IDs of type int16\_t in the released games. This is signed 16-bit value.

When *fastman92 limit adjuster* got more advanced, it brought a possibility of uint16\_t ID values – unsigned 16-bit values.

Later it was seen that the unsigned 16-bit IDs were still limiting the modding and something had to be done about it.

int32\_t values, which are stored externally have been introduced.

It’s recommended that you will use a **CFileID** in game structures of your project, instead of int16\_t / uint16\_t for the file ID members, to make sure they can support 32-bit values.

### CFileID

**CFileID** is the class made specifically to provide a support to access the ID values stored in the structures, which in default game would always be of type int16\_t.

Some features of CFileID:

* sizeof(CFileID) is 2. It still takes as much place as an old 16-bit value, when placed in the structure.
* Works without the FLA.
* Will access 16-bit ID, if 32-bit one is not possible.
* Will access 32-bit ID, if possible.
* Overloading operators are provided, so that CFileID can act still act like an old integral type of 16-bit ID, when it comes to C++ syntax. For example, a statement   
  pEntity->modelID = 400;

is still possible due to assignment = operator being implemented.

* You don’t have to be sure if specific member is patched by the FLA to use with the possibility to use 32-bit or not. At worst, the 16-bit value will be accessed. Your plugin will not crash.

### How to integrate CFileID into structures

The type **CFileID** should be used to access the file/model/texture archive IDs, which were previously always a signed 16-bit value.

At first you need include a file, which declares the CFileID type:

#include <FLA\_not\_required/IDaccess.h>

If you still haven’t read how to integrate the **FLA\_not\_required** into your project, do it first: Use FLA\_not\_required

Suppose you have this structure:

class CStreamingInfo

{

public:

int16\_t usNext; // see, it's a file ID!

int16\_t usPrev; // see, it's a file ID!

int16\_t usNextOnCd; // see, it's a file ID!

char ucFlags;

uint8\_t ucImgId;

int iBlockOffset; // m\_OriginalSizeInSectors

int iBlockCount;

uint8\_t uiLoadFlag;

};

Now this has a problem, that IDs are limited to what int16\_t offers. See the table: Table of possible IDs

The FLA can offer unsigned 16-bit IDs though:

class CStreamingInfo

{

public:

uint16\_t usNext; // see, it's a file ID!

uint16\_t usPrev; // see, it's a file ID!

uint16\_t usNextOnCd; // see, it's a file ID!

char ucFlags;

uint8\_t ucImgId;

int iBlockOffset; // m\_OriginalSizeInSectors

int iBlockCount;

uint8\_t uiLoadFlag;

};

But this still has a problem, of the IDs being limited to what uint16\_t offers and this is not so much. See the table: Table of possible IDs

We’re going to use CFileID use for the best possibilities:

class CStreamingInfo

{

public:

CFileID usNext; // see, it's a file ID!

CFileID usPrev; // see, it's a file ID!

CFileID usNextOnCd; // see, it's a file ID!

char ucFlags;

uint8\_t ucImgId;

int iBlockOffset; // m\_OriginalSizeInSectors

int iBlockCount;

uint8\_t uiLoadFlag;

};

Now these members using CFileID will allow for a compatibility with 16-bit or 32-bit depending on what the game is using. This is highly recommended!

With the type of file ID members in CStreamingInfo we have solved a problem of compatibility with the count of file IDs significiantly higher than in standard game.

Note that your project doesn’t really have to use a CStreamingInfo structure. If it doesn’t have a CStreamingInfo structure, then you don’t have to add it, it’s apparently unneccessary for your project.

The major step of the FLA development was an introduction of 32-bit difficult IDs – model and texture archive IDs.

There are many structures, which also have the model IDs or texture archive IDs. You need to update these structures that your project uses.

Example:

class CEntity : public CPlaceable

{

public:

// ... some members ...

CFileID modelID; // offset +0x22

// ... some members ...

};

### List of structures containing 16-bit model ID

If you use any of these structures, change the type of the model IDs to CFileID for the specified members.

1. CBaseModelInfo

* int16\_t m\_nTxdIndex;

WIN\_X86 GTA\_SA offset: 0xA

1. CTimeInfo

* int16\_t m\_wOtherTimeModel;

WIN\_X86 GTA\_SA offset: 0x2

1. CCarGenerator, used in save file
   * int16\_t m\_wModelIndex;

WIN\_X86 GTA\_SA offset: 0x22

1. CCollisionFile
   * int16\_t m\_wLowestModelID;

WIN\_X86 GTA\_SA offset: 0x22

* + int16\_t m\_wHighestModelID;

WIN\_X86 GTA\_SA offset: 0x24

1. CEntity
   * int16\_t m\_wModelIndex;

WIN\_X86 GTA\_SA offset: 0x22

1. CIplFile
   * int16\_t m\_usBuildingsBegin;

WIN\_X86 GTA\_SA offset: 0x22

* + int16\_t m\_usBuildingsEnd;

WIN\_X86 GTA\_SA offset: 0x24

* + int16\_t m\_usDummyBegin;

WIN\_X86 GTA\_SA offset: 0x26

* + int16\_t m\_usDummyEnd;

WIN\_X86 GTA\_SA offset: 0x28

1. CObject
   * int16\_t m\_nRefModelIndex;

WIN\_X86 GTA\_SA offset: 0x14A

* + int16\_t m\_wRemapTxd;

WIN\_X86 GTA\_SA offset: 0x16A

1. CStoredCar, used in save file
   * int16\_t m\_wModelIndex;

WIN\_X86 GTA\_SA offset: 0x12

1. CPickup, used in save file
   * int16\_t m\_wModelIndex;

WIN\_X86 GTA\_SA offset: 0x18

1. CScriptForBrain
   * int16\_t m\_wModelIndex;

WIN\_X86 GTA\_SA offset: 0xC

1. CVehicle
   * int16\_t m\_wPreviousRemapTxd;

WIN\_X86 GTA\_SA offset: 0x598

* + int16\_t m\_wRemapTxd;

WIN\_X86 GTA\_SA offset: 0x59A

1. Furniture\_c
   * uint16\_t m\_wModelIndex;

WIN\_X86 GTA\_SA offset: 0x8

1. CWaterCreatureInfo
   * uint16\_t\* pModelIndex;

WIN\_X86 GTA\_SA offset: 0x0

1. uint16\_t\* CVehicleModelInfo\_\_ms\_upgradeWheels;
2. RW plugin 0x253F200, atomic
   1. int16\_t m\_wModelIndex;

WIN\_X86 GTA\_SA offset: 0x0

### List of structures containing 16-bit TXD ID

If you use any of these structures, change the type of the model IDs to CFileID for the specified members.

1. CVehicleModelInfo
   * int16\_t PaintjobTxdArray[4];

WIN\_X86 GTA\_SA offset: 0x18

1. defaultTXDs, see CTxdStore::Initialise

### File info streaming

#### CStreaming::ms\_aInfoForModel

The games GTA III, GTA Vice City and GTA San Andreas and Bully Scholarship Edition use an array to describe the state of the files from the IMG archives:

CStreamingInfo ms\_aInfoForModel[n];

Where n is the max number of elements.

In case of GTA III, GTA Vice City and GTA San Andreas this is exactly the count of all file IDs.

In Bully Scholarship Edition, it’s a bit less than count of all IDs. In standard Bully SE it’s the count of IDs, which will really be used and a bit extra.

You can no longer assume that an array is still placed on the old default memory address.

It’s necessary to read the possibly updated address.

The following addresses may be used to read a pointer to CStreaming::ms\_aInfoForModel

* **WIN\_X86**
  + Bully Scholarship Edition 1.20 – (0x5334DF + 6)
  + GTA III v1.0 EXE – (0x409757 + 4)
  + GTA Vice City v1.0 - (0x40FFB5 + 1)
  + GTA San Andreas v1.0 compact/EXE HOODLUM - (0x5B8B08 + 6)
  + GTA IV v1.0.4.0 EXE – (0x873B64 + 3)
  + GTA IV v1.0.7.0 EXE – (0x9884A4 + 3)
  + GTA IV v1.0.8.0 EXE – (0x8DAC74 + 3)
  + GTA EFLC v1.1.2.0 EXE – (0x9BA534 + 3)
  + GTA EFLC v1.1.3.0 – (0x8A5784 + 3)

#### Base IDs

The games GTA III, GTA Vice City and GTA San Andreas and Bully Scholarship Edition use the base IDs, which relate to file streaming, for every different file type.

For example, in GTA SA TXD base index is 20000.

The following addresses store 32-bit values and may be used to read the base IDs for the file streaming:

* Model archive has index of 0
* Texture archive
  + WIN\_X86
    - Bully Scholarship Edition 1.20 – (0x5334DA + 1)
    - GTA III v1.0 EXE - (0x40A80F+ 1)
    - GTA Vice City v1.0 - (0x61DC3F + 2)
    - GTA San Andreas v1.0 compact/EXE HOODLUM – (0x407104 + 2)
* Collision archive
  + WIN\_X86
    - Bully Scholarship Edition 1.20 – (0x52DFF4 + 1)
    - GTA Vice City v1.0 - (0x629FB8 + 1)
    - GTA San Andreas v1.0 compact/EXE HOODLUM - (0x410344 + 2)
* Item placement
  + WIN\_X86
    - Bully Scholarship Edition 1.20 - (0x52E3B7 + 1)
    - GTA San Andreas v1.0 compact/EXE HOODLUM - (0x4044F4 + 2)
* Path nodes
  + WIN\_X86
    - GTA San Andreas v1.0 compact/EXE HOODLUM - (0x44D064 + 2)
* Animation archive
  + WIN\_X86
    - GTA Vice City v1.0 - (0x404969 + 2)
    - GTA San Andreas v1.0 compact/EXE HOODLUM - (0x407124 + 2)
* Car recording
  + WIN\_X86
    - GTA San Andreas v1.0 compact/EXE HOODLUM – (0x4594A1 + 2)
* SCM script
  + WIN\_X86
    - GTA San Andreas v1.0 compact/EXE HOODLUM - (0x46A574 + 2)

If you need to know the value of ID limit, you should do the subtraction.

Example:

numberOfTXDfiles = baseCOLindex - baseTXDindex;

To know the limit of SCM scripts in GTA San Andreas v1.0 compact/EXE HOODLUM, you should read uint16\_t value from (0x470917 + 2)

#### Type of file IDs

### Individual types

#### Model ID limit

If the model ID limit is increased, the following array from the class CModelInfo will be reallocated:

CBaseModelInfo\*ms\_modelInfoPtrs[n];

Where n is the max number of elements, the model ID limit, for example 20000

Because you have a library, which is no longer a part of game source code itself, you will have it declared as:

CBaseModelInfo\*\*ms\_modelInfoPtrs;

CBaseModelInfo is the structure, which all objects, which define models in IDE files should use.

You can no longer assume that an array is still placed on the old default memory address.

It’s necessary to read the possibly updated address.

The following addresses may be used to read a pointer to CModelInfo::ms\_modelInfoPtrs

* **WIN\_X86**
  + Bully Scholarship Edition 1.20 - (0x5359D4 + 3)
  + GTA III v1.0 EXE – (0x40A804 + 3)
  + GTA Vice City v1.0 - (0x55F7E0 + 3)
  + GTA San Andreas v1.0 compact/EXE HOODLUM – (0x403DA4 + 3)
* **ANDROID\_ARM32**
  + GTA SA 2.00 – (0x6796D4)

Example, how to read the address of **ms\_modelInfoPtrs** for GTA SA in your library:

CModelInfo::ms\_modelInfoPtrs = \*(CBaseModelInfo\*\*\*)(0x403DA4 + 3);

#### Texture archive ID limit

This one uses a pool CTxdStore::ms\_pTxdPool, which gets allocated with the specified number of possible elements.

## uint8\_replacement

**uint8\_replacement** is the class made specifically to provide a support to access the values stored in the structures, which in default game would always be of type uint8\_t.

Some features of uint8\_replacement:

* sizeof(uint8\_replacement) is 1. It still takes as much place as an old 8-bit value, when placed in the structure.
* Works without the FLA.
* Will access 8-bit ID, if 32-bit one is not possible.
* Will access 32-bit ID, if possible.
* Overloading operators are provided, so that uint8\_replacement can act still act like an old integral type of 8-bit ID, when it comes to C++ syntax. For example, a statement   
  pColModel->COL\_file\_ID = COL\_ID;

is still possible due to assignment = operator being implemented.

* You don’t have to be sure if specific member is patched by the FLA to use with the possibility to use 32-bit or not. At worst, the 8-bit value will be accessed. Your plugin will not crash.

The type **uint8\_replacement** should be used to access the values, which were previously always a 8-bit value and which can be potentially externalized by the FLA.

At first you need include a file, which declares the uint8\_replacement type:

#include <FLA\_not\_required/IDaccess.h>

After that, you should alter the types of structure members, which may use a uint8\_replacement type with the FLA, an example might be [the extended weapon ID](#_Extended_weapon_ID).

## Shadow limits

### Real time shadows

Real time shadows are used for shadows of objects, peds and vehicles.

The game internally uses the following variable:

CRealTimeShadowManager g\_realTimeShadowMan;

This is how class CRealTimeShadowManager looks by default:

class CRealTimeShadowManager

{

bool m\_initialized;

bool field\_1;

CRealTimeShadow\* m\_realTimeShadowPool[16];

CShadowCamera m\_camera1;

CShadowCamera m\_camera2;

};

VALIDATE\_SIZE\_ONLY\_ON\_32\_BIT\_ARCHITECTURE(CRealTimeShadowManager, 0x54);

With the FLA the class CRealTimeShadowManager may be reallocated. This structure is now dynamic.

The offset to member m\_realTimeShadowPool may be different.

The number of elements for an array m\_realTimeShadowPool may be different.

The variable g\_realTimeShadowMan may be relocated.

#### g\_realTimeShadowMan

This variable from game EXE may be relocated and pointer should be read.

* **WIN\_X86**
  + GTA San Andreas v1.0 compact/EXE HOODLUM - (0x53BE62 + 1)

#### Offset to m\_realTimeShadowPool from CRealTimeShadowManager

If you want to retrieve an offset of m\_realTimeShadowPool from CRealTimeShadowManager, you can use the following exported function:

// Returns offset of CRealTimeShadowManager::m\_realTimeShadowPool

F92\_LA\_API unsigned int GetOffset\_CRealTimeShadowManager\_m\_realTimeShadowPool();

## Number of elements in m\_realTimeShadowPool from CRealTimeShadowManager

The number of elements in m\_realTimeShadowPool from CRealTimeShadowManager directly depends on the limit of Real time shadows configured for the FLA.

The following address stores a 32-bit value, the value of Real time shadows:

* **WIN\_X86**
  + GTA San Andreas v1.0 compact/EXE HOODLUM - (0x706B7F + 1)

## Weapon limits

### Extended weapon ID

If necessary (the limit of weapon IDs is set to value higher than what unsigned 8-bit value would handle), the weapon ID will become 32-bit.

The following member of the structure CVehicle is the subject to change:

uint8\_replacement m\_nLastWeaponDamageType; // offset +0x508 in GTA San Andreas

### Parent weapon ID

The added weapons typically use a parent weapon ID, which specifies the internal functionality of the weapon. You may want to retrieve the parent weapon ID in your plugin/script.

For that an exported function from the FLA has been created:

// Returns weapon parent type

F92\_LA\_API int F92\_LA\_CONVENTION GetWeaponHighestParentType(int weaponType);

See the file Exports.h

### Weapon name

The added weapons have a name defined in config file. If you want to want to retrieve a name for a specified weapon ID you can use this exported function:

// Returns weapon name

// Can only return a name, if weapon limits are enabled and weapon config DAT got loaded.

F92\_LA\_API const char\* F92\_LA\_CONVENTION GetWeaponName(int weaponType);

### Count of weapon infos

The count of weapon infos relates to the number of possible elements in an array aWeaponInfo.

The count depends on the current configuration of weapons with the file that came with the FLA.

To get a count of weapon infos you should use the following exported function:

// Returns the count of weapon infos

F92\_LA\_API unsigned int F92\_LA\_CONVENTION GetCountOfWeaponInfos();

### Number of weapon types

The number of weapon types is the limit which specifies how many weapon IDs may be defined.

To get a limit value of weapon types you can use the following exported function:

// Returns the number of possible weapon types

F92\_LA\_API unsigned int F92\_LA\_CONVENTION GetNumberOfWeaponTypes();

## Vehicle colors

Vehicle colors is the limit which specifies how many colors may be defined inside of carcols.dat

### Reallocated color array

In game, there’s an array like this:

class CVehicleModelInfo

{

public:

static CColorType ms\_vehicleColourTable[NUMBER\_OF\_VEHICLE\_COLOURS];

};

Where:

* CColorType – this is the type used to store the color value. This type may be different between games:
  + CRGBA - in GTA San Andreas
  + CBGRA - in GTA IV
* NUMBER\_OF\_VEHICLE\_COLOURS - the maximum number of elements in an array, the limit value

In order to increase the limit of vehicle colors this array gets reallocated by the FLA.

If you want to access this array in your project, you’ll need to read a pointer to it first. The following addresses may be used to read a pointer to CVehicleModelInfo::ms\_vehicleColourTable:

* **WIN\_X86**
  + GTA San Andreas v1.0 compact/EXE HOODLUM - (0x447094 + 3)

### Extended weapon ID

If necessary (the limit of weapon IDs is set to value higher than what unsigned 8-bit value would handle), the weapon ID will become 32-bit.

If you use any of these structures, change the type of the color IDs to uint8\_replacement for the specified members.

1. CObject
   * uint8\_t m\_nVehiclePartPrimaryColor;

WIN\_X86 GTA\_SA offset: 0x14C

* + uint8\_t m\_nVehiclePartSecondaryColor;

WIN\_X86 GTA\_SA offset: 0x14D

* + uint8\_t m\_nVehiclePartTertiaryColor;

WIN\_X86 GTA\_SA offset: 0x14E

* + uint8\_t m\_nVehiclePartQuaternaryColor;

WIN\_X86 GTA\_SA offset: 0x14F

1. CVehicle
   * uint8\_t m\_nPrimaryColor;

WIN\_X86 GTA\_SA offset: 0x434

* + uint8\_t m\_nSecondaryColor;

WIN\_X86 GTA\_SA offset: 0x435

* + uint8\_t m\_nTertiaryColor;

WIN\_X86 GTA\_SA offset: 0x436

* + uint8\_t m\_nQuaternaryColor;

WIN\_X86 GTA\_SA offset: 0x437

1. CVehicleModelInfo
   * uint8\_t m\_anPrimaryColors[8];

WIN\_X86 GTA\_SA offset: 0x2B0

* + uint8\_t m\_anSecondaryColors[8];

WIN\_X86 GTA\_SA offset: 0x2B8

* + uint8\_t m\_anTertiaryColors[8];

WIN\_X86 GTA\_SA offset: 0x2C0

* + uint8\_t m\_anQuaternaryColors[8];

WIN\_X86 GTA\_SA offset: 0x2C8

* + uint8\_t m\_nCurrentPrimaryColor;

WIN\_X86 GTA\_SA offset: 0x2D2

* + uint8\_t m\_nCurrentSecondaryColor;

WIN\_X86 GTA\_SA offset: 0x2D3

* + uint8\_t m\_nCurrentTertiaryColor;

WIN\_X86 GTA\_SA offset: 0x2D4

* + uint8\_t m\_nCurrentQuaternaryColor;

WIN\_X86 GTA\_SA offset: 0x2D5

1. uint8\_t PrimaryColor;
   * GTA San Andreas v1.0 compact/EXE HOODLUM - (0xB4E3F0)
2. uint8\_t SecondaryColor;
   * GTA San Andreas v1.0 compact/EXE HOODLUM - (0xB4E3F1)
3. uint8\_t TertiaryColor;
   * GTA San Andreas v1.0 compact/EXE HOODLUM - (0xB4E3F2)
4. uint8\_t QuaternaryColor;
   * GTA San Andreas v1.0 compact/EXE HOODLUM - (0xB4E3F3)

# Adding vehicles to GTA / Bully

Adding a vehicle is easy.  
  
Limits to set up:

* [IDE LIMITS]
  + ; IDE : cars section (212)

#Vehicle Models = 212  
  
Total number of IDE entries that can be defined by ‘cars’ section.

* [HANDLING.CFG LIMITS]
  + #Apply handling.cfg patch = 0

If 0, the FLA will consider handling.cfg limits to be disabled completely.

* + ; Number of standard lines (210)

#Number of standard lines = 210  
The handling requires a standard line as a minimum.

* + ; Number of bike lines (13)

#Number of bike lines = 13  
Bike handlings also require a bike line, proceeded with the **!** character.

* + ; Number of flying lines (24)

#Number of flying lines = 24

Flying handlings (for helicopters and planes) also requires a flying line, preceded with the **$** character.

* + ; Number of boat lines (12)

#Number of boat lines = 12

Boat handlings also require a bike line, preceded with the **%** character.

* [ADDONS]
  + #Enable vehicle audio loader = 0

The FLA offers a vehicle audio loader, which allows to set sound parameters for vehicles, by identified by model name in configuration file.

* + #Enable FXT loader = 0

FXT loader allows for creating FXT file with new texts, which may include the names of vehicles.

For GTA 3, GTA VC and GTA SA the FLA does not offer FXT loader, because of CLEO, which offers one. For these games use CLEO’s FXT loader instead.

Absolutely necessary:

* IMG archive – add model and texture archive files of vehicle.
* IDE file (typically vehicles.ide) – add IDE entry, model name and texture archive name must correspond with files added.  
  If you’re adding a vehicle to game earlier than GTA IV, it uses statically defined IDs. In such a case you must use an ID for a vehicle, which is free – it’s not used by any other IDE entry. The ID must be within a range of model ID limit / DFF ID limit.

Optional:

* handling.cfg / handling.dat – you may want to add the handling lines, which tell the game, how the physics of the vehicle should work.

The handling may consist of the following lines:

* standard line
* bike line preceded with $ sign.
* flying line
* boat line

For games before GTA IV, the list of handling names is hardcoded in game EXE.   
This means that unless the FLA has an option to increase the handling.cfg limits for such a game, you can’t add any new entries to handling.cfg.  
The FLA has got a handling.cfg limits patched for GTA VC 1.0 WIN\_X86 and GTA SA 1.0 WIN\_X86.

* + - The handling identifier is assigned to vehicle using one of fields in IDE line – HandlingId.  
        
      Take a look, GTA SA example:
    - # Id, Model name, Txd name, Type **HandlingId** Game name, Anims Class Frq flags Comprules
    - #
    - # cars have two extra fields wheelmodelId and wheel scale
    - # planes have one extra field model id of low level of detail
    - 400, landstal, landstal, car, **LANDSTAL**, LANDSTK, null, normal, 10, 0, 0, -1, 0.768, 0.768, 0
* carcols.dat – it’s not necessary to edit this file, but if you don’t then the vehicle spawned will be black.  
  Add an entry, using the model name as defined in IDE file.
* carmods.dat – should the car should be tunable, you must edit this file. Add an entry, using the model name as defined in IDE file.
* Vehicle audio settings – this is file, which comes with the FLA (for example GTA SA: gtasa\_vehicleAudioSettings.cfg), which works, if vehicle audio loader is enabled.  
  The sound lines are assigned to specified vehicles by their model names from IDE.
* GXT / FXT entry

if you want a vehicle to have some name I recommend creating FXT file, this is text file with FXT extension, for example vehicles.fxt, put it in CLEO\CLEO\_TEXT if CLEO’s FXT loader is used or FXT directory, which came with the FLA.  
Create vehicles.fxt like this:

AT400 Boeing 737 LOT

ELEGY BMW M3

HOTKNIF Talbot Lago T26C

PCJ600 Aprilia RSV4

NRG500 Suzuki GSX-R600

WAYFARE Hexer

SANCHEZ Yamaha YZ250F

The game name identifier is assigned to vehicle using one of fields in IDE line – Game name.  
  
Take a look, GTA SA example:

# Id, Model name, Txd name, Type HandlingId **Game name**, Anims Class Frq flags Comprules

#

# cars have two extra fields wheelmodelId and wheel scale

# planes have one extra field model id of low level of detail

400, landstal, landstal, car, LANDSTAL, **LANDSTK**, null, normal, 10, 0, 0, -1, 0.768, 0.768, 0

* Model special features (optional step).  
  Certain standard vehicle models have special features, for example hydra – VTOL and sound.

There’s a good tutorial on adding new vehicles written by **Junior\_Djjr**, which can be accessed on the following address:

<https://www.mixmods.com.br/2015/12/tutorial-adicionar-carros-sem-substituir.html>

# Adding weapons to GTA

There’s a good tutorial on adding new weapons written by **Junior\_Djjr**, which can be accessed on the following address:

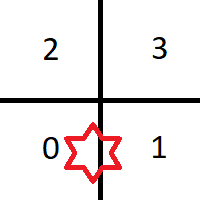
<https://www.mixmods.com.br/2016/05/tutorial-adicionar-armas-sem-substituir.html>

# Explanation of world sectors

Game may a use a concept of world sectors to keep a track of entities at certain areas on the map.  
The map uses a grid and divides a map into blocks of equal size.

The entity may occupy more than one world sector depending on its position and dimensions.

This happens in the situation when an entity is placed across the areas of the sectors:



In the example, the entity (star on the illustration) occupies two world sectors: sector 0 and sector 1.

## How map is divided into sectors

Let’s examine how it works.

Default GTA SA world map size is 6000 x 6000.

Map coordinates of X and Y are in range -3000.0 to 3000.0

Block size of world sectors is 50 in game units.

Number of sectors per one dimension will be 6000 (map size) / 50 = 120.

Number of sectors per two dimensions will be a result of multiplication of number of sectors per X dimension (120) and number of sectors per Y dimension (120 as well).

Number of sectors in two dimensions will be 120 \* 120 = 14400.

Let’s calculate the ID of world sector that will be used for the following coordinates:

Coord X = 2300.0

Coord Y = -650.0

ID of world sector X = (our X coordinate - start of map for X) / world sector size

ID of world sector X = (2300.0 + 3000.0) / 50.0 = **106**

ID of world sector Y = (our Y coordinate - start of map for Y) / world sector size

ID of world sector Y = (650.0 + 3000.0) / 50.0 = **73**

World sector ID = ID of world sector Y \* Number of world sectors per dimension + ID of world sector X

World sector ID = 73 \* 120 + 106 = **8866**

## World sectors

These are used for the following types of entities:

* Buildings
* Dummies

Game uses single linked list for these sectors.

The game can traverse the list in one direction only.

## World repeat sectors

These are used for the following types of entities:

* Vehicles
* Peds
* Objects
* Portal instances (in GTA IV)
* Interior instances (in GTA IV)

Game uses double linked list for these sectors.

World repeat sector ID per single dimension is calculated taking world sector ID per dimension and applying modulo of 16 to it.

Note that the following operations yield the same result:

*x & 0xF = x % 0x10*

where *x* is an integer number.

Number of repeat sectors per dimension = 16

World repeat sector ID = ID of world sector Y & 0xF \* Number of repeat sectors per dimension + ID of world sector X & 0xF

Disclaimer: Bully Scholarship Edition does not use world repeat sectors. Lists of vehicles, peds and objects are the part of the world sectors and they’re single linked lists.

## World LOD sectors

These are used for the LOD entities.

Bully Scholarship Edition does not use world LOD sectors.