Getting started with Assetto Corsa application development

Starting with Assetto Corsa application development is not hard, but it is harder than it needs to be. The common wisdom seems to be to read the sample applications that come with Assetto Corsa (in /apps/python/, there is a Chat and gMeter application), or to read the code behind some of the applications being shared on the forums.

This certainly works, and is how I learned, but it takes more guess-and-test than should be necessary. For example, early on in reading the forum I heard of something called py_log.txt, but it wasn't clear where to find it.

In this document, I pull back the curtain and explicitly show the basics. I assume you already know some Python. Other than a small note about Python's treatment of global variables, I do not cover any Python.

Often, I will type AC as shorthand for Assetto Corsa.

I'm going to develop below an application called appName. You should, of course, name your application something more descriptive.

Preliminaries

Locations of interest

There are two folders you should be aware of:

- The Assetto Corsa installation directory. This is most likely to be found in your Steam directory. On Windows, this look like C:\Program Files (x86)\Steam\steamapps\common\assettocorsa. You may have chosen to install AC elsewhere, and I trust you can find where that is.
- The Assetto Corsa documents directory. On Windows, this looks like C:\User\My Documents\Assetto Corsa\, where Useris replaced with your Windows account name.

Each of these contains at least one subfolder of interest:

- In the installation directory, the interesting subfolder is /apps/. This is where the application is placed.
- In the documents directory, the subfolder of interest is /logs/. In this directory you will find both log.txt and py_log.txt.

log.txt is where AC logs everything about the execution of AC itself. Sometime this will contain relevant information about your application that AC logs automatically. py_log.txt is where AC places strings explicitly requested to be logged by running applications.

Basic workflow

The workflow of testing an application isn't the best. It can be slow going when you're making a lot of changes, especially if you make syntax or logical mistakes. Try to be careful and ensure the code you're attempting to run is correct. You might want to look

into something like pylint so you can find errors in the code without having to run Assetto Corsa.

If something is wrong with your code, and error message might show up automatically in the in-game console or inpy_log.txt. It will always show up in log.txt. The best way to find an error in log.txt is by search for your application name, in this case appName, and reading the surrounding output.

I call an on-track event a session. I usually test in practise mode around a short track like silverstone-international, but it shouldn't matter what you choose.

Here is how I test my application:

- I edit the source code, then run AC and start a session.
- If there are no errors and the application was previously active on the screen, it will still be so. If something went wrong, it will have disappeared and there will be a message somewhere as to why. If an error has occurred, I end the session. Otherwise, I continue.
- I do what I need to in order to test the application behaviour.
- When I find that I need to make changes, I end the session.

In either case - an error or a desired change - you don't have to exit Assetto Corsa. Instead, you can alt-tab out, change the code, and then start a new session. This is faster than continuously starting and exiting the main Assetto Corsa application. It's still a bit of a drag having to exit and restart sessions, so as I noted before try to be careful that at least the syntax of your code is correct before testing it. Otherwise, you wait around while starting a session only to find out your application has failed to load.

It's a habit of mine to always check the console at the start of a session. If something in my application is wrong, there is likely to be a message in the console about what went wrong. It's also often the case that the message in the console contains the line number in my application where the error was found. If this is not the case, the error might instead be in py_log.txt orlog.txt. Again, you have a good change of finding a specific line number there, or at worst a helpful error message.

Getting an application running

Create a folder in /apps/python/ with the name appname. Inside /apps/python/appname/, create a file appname.py. Open the file for editing.

A most basic application

First, note that a barebones application still needs a few imports. I won't keep embedding these in the code snippets below, so be aware that every application should start off with the following imports:

Code:

```
import sys
import ac
import acsys
```

The most basic applications only takes a few lines of code. The AC plugin architecture will automatically execute certain functions in which it expects to find your code. To begin an application, you must define a function as follows:

Code:
```
def acMain(ac_version):
...
```
The code for your application will go in place of the ellipses. For now, we'll insert the bare minimum code:
Code:
```
def acMain(ac_version):
    appWindow = ac.newApp("appName")
    ac.setSize(appWindow, 200, 200)
    return "appName"
```
Actually, the bare minimum is probably just the return statement, but that application is not at all interesting.

If you run Assetto Corsa and start a session, you will find in the application sidebar an entry named appName. If you activate this, you will see a very basic widget consisting of a 200x200 application window with the name appName at the top. Here is what you should see in the application sidebar:

ac.log and ac.console
The function ac.log writes to the file py_log.txt which I mentioned earlier.

The function ac.console writes to the Assetto Corsa console. To bring up the console, hit the Home key on your keyboard. Hit the key again to dismiss the console.

The way you might use these functions is quite similar, so think of it this way:

- Use ac.log when you want the text to persist after the session has ended. This is helpful if you want to debug the application through lots of printed statements.
- Use ac.console when you might want to read the output during the session. By bringing up the in-game console you can immediately view the messages. Yes, you can alt-tab out and view the py_log.txt while the session is still running, but it's not nearly as pleasant.

These functions are both good targets to dump information that you're not quite sure about. Use them to figure out exactly that a piece of code is doing.

Extending the basic application
Let's change the code to the following:
Code:
```
def acMain(ac_version):
   appWindow = ac.newApp("appName")
   ac.setSize(appWindow, 200, 200)

   ac.log("Hello, Assetto Corsa application world!")
   ac.console("Hello, Assetto Corsa console!")
   return "appName"
```
Start a new session, and check that you see the text Hello, Assetto Corsa console! in the console when you hit the Homekey on your keyboard. Additionally, ensure that Hello, Assetto Corsa application world! has shown up in the filepy_log.txt.

Unsurprisingly, both should be the case. There were no tricks here. One important thing to note is that your application does not have exclusive usage of either the console or python log file. Other applications you have installed might also be sending text to the console or python log. You can either disable all other application, or prefix all message with a unique string, e.g.*** Message from appName: so that you can quickly find the output from your application.

Adding labels to your application window
If you have an application window on your screen, you probably want to display some information within it. To do so, we can add labels to the window:
Code:
```
   l_lapcount = ac.addLabel(appWindow, "Laps: 0");
   ac.setPosition(l_lapcount, 3, 30)
```
Remember, your application windows is a 200x200 widget. Some of this space is taken up by the header, where the appNamelabel automatically appears. This is why I set the label at position 30 vertically. I set it at 3 horizontally to offset it slightly from the border.

The code should now look like this:

Code:
```
def acMain(ac_version):
   appWindow = ac.newApp("appName")
   ac.setSize(appWindow, 200, 200)

   ac.log("Hello, Assetto Corsa application world!")
   ac.console("Hello, Assetto Corsa console!")

   l_lapcount = ac.addLabel(appWindow, "Laps: 0");
   ac.setPosition(l_lapcount, 3, 30)
   return "appName"
```
and the application window should look like this:

Moving towards a more realistic application
So far, our application consists only of the application window and a static label. Let's add some dynamic behaviour.

The function acMain has setup our application window. To do something with it, we must use an additional functionacUpdate.

One important thing to note is that we're going to need to access the label l_lapcount from within acUpdate if we want to place dynamic information into it. So far, the label has been a variable local to acMain. Since we're not the one callingacUpdate, we can't pass the label along to it as a parameter. Instead, we must make l_lapcount a global variable. To do so, define it outside of acMain. Then, within acMain we must inform the function that l_lapcount is a global variable. If we forget to do so, we'll create a local variable l_lapcount within acMain which will shadow the global variable, and any changes we make within acMain will not be visible outside of it. Most importantly, if we forget to do this the actual label we placed in the application window in acUpdate would not be available from acUpdate.

We'll also add a global variable lapcount which only needs to be accessible within acUpdate. The code should look like so:

Code:
```
l_lapcount=0
lapcount=0

def acMain(ac_version):
   global l_lapcount

   appWindow = ac.newApp("appName")
   ac.setSize(appWindow, 200, 200)

   ac.log("Hello, Assetto Corsa application world!")
   ac.console("Hello, Assetto Corsa console!")

   l_lapcount = ac.addLabel(appWindow, "Laps: 0");
   ac.setPosition(l_lapcount, 3, 30)
   return "appName"

def acUpdate(deltaT):
   global l_lapcount, lapcount
   laps = ac.getCarState(0, acsys.CS.LapCount)
   if laps > lapcount:
      lapcount = laps
      ac.setText(l_lapcount, "Laps: {}".format(lapcount))
```
acUpdate takes a parameter that is ???(Guess: ?milli?seconds since it was called last).

Note that within acUpdate we make a call ac.getCarState(0, acsys.CS.LapCount). This might look confusing at first since I never explained anything about it, but it's just another function made available through our import of ac and acsys. I don't want to duplicate the official documentation, so please look at the resources section at the end of this guide for a link to the official documentation. Eventually, you should read it so that you know what has been made available for application development by the Assetto Corsa developers, but it's not important at the moment. You can continue on with this guide.

Now, after completing a lap your application window will look like this:

and so on as you complete laps.

You could, of course, also log this information to the console or the python log:

Code:
```
   ac.log("{} laps completed".format(lapcount))
   ac.console("{} laps completed".format(lapcount))
```
Additional functions called by Assetto Corsa

One additional function to note is acShutdown, which is called when the session is ended.

Code:
```
def acShutdown():
   # ...
   return
```

You'll want to add within acShutdown any code that should be completed before your application exits. For example, if there are outstanding database modifications, you want to make sure you commit them and safely close the connection to the database.

You can also register callbacks for certain events. Two that I am aware of are ac.addOnAppActivatedListener andac.addOnAppDismissedListener. For instance, you might define a function on_activation and register it by calling

Code:
```
ac.addOnAppActivatedListener(appWindow, on_activation)
```
It seems that acUpdate is always called, even when the application is dismissed. If this is not desirable, the idiom would be to check a flag within acUpdate that is set in the callback registered with ac.addOnAppActivatedListener, so that you only run code when the application is activated.

Accessing Shared Memory

The Python API gives us access to some nice stuff, but there is a lot more than you might need access to in order to make your app. To get to this additional information, you must access the shared memory structure made available by Assetto Corsa. Here is a quick tutorial on doing so:

- Add a directory within /apps/python/appname/ with a name of your choice. I use third_party.
- Add into this directory _ctypes.pyd and sim_info.py. See Shared memory for Python applications (sim_info.py) for AC v0.20 for where to obtain these files.
- Insert the third_party directory into the python environment before using the import statement: sys.path.insert(len(sys.path), 'apps/python/appname/third_party').
- Import from sim_info.py using from sim_info import info.

After doing so, you can get to any of the shared memory information using e.g. info.physics.fuel.

Exercise: Add an additional label to the application window and fill it with the current fuel in the tank. Update the value dynamically throughout the session with a period shorter than once-per-lap.

If you can complete this, you have understood everything I tried to communicate by writing this document.

Enough, for now

That should cover the basics, and get you started writing Assetto Corsa applications. There is still a lot that you'll need to learn to make a non-trivial application, but a good starting point is always helpful.
Also, please note that this is just one way to do it. In programming, like in everything else there are many solution to the same problem, so feel free to experiment your own ideas

once you get the hang of it. There is nothing you could break .

Ressources

Assetto Corsa Python documentation
Assetto Corsa Shared Memory Reference