ChatScript Advanced Layers Manual

Revision 7/18/2021 cs11.5

LAYERS

ChatScript loads its startup data layers at a time.

Layer 0 comes from :build 0 typically, which is common stuff that comes with ChatScript. You can override its contents by providing your own files0.txt file. The layer is intended for stuff that won't be changing much (thus saving recompilation time). Some people have used layer 0 to house their entire bot, with layer 1 used for downloadable patches on phone apps.

Layer 1 comes from :build harry or :build yourbot and is typically your bot.

Layer 2 is a boot layer, which may not have any content. More later.

Layer 3 is the user layer. Every volley starts out by loading it from the user's topic file. At the end of the volley data is written back out to that file, and this layer is removed.

BOOT layer

The boot layer is used for adding dynamic data to the basic server on startup (since layers 0 and 1 are precompiled).

`^CSBOOT()`

outputmacro: ^CSBOOT()

This function, if defined by you, will be executed on startup of the ChatScript system. It is a way to dynamically add facts and user variables into the base system common to all users (the boot layer). And returned output will go to the console and if a server, into the server log Note that when a user alters a system `$variable`, it will be refreshed back to its original value for each user.

If you create JSON data, you should probably use ^jsonlabel() to create unique names separate from the normal json naming space.

Changing the content of boot json facts (e.g. `$data.value += 1`) may create abandoned data in the boot layer (the old value of $data.value) and do this often enough and you may run out of memory since there is no way to reclaim it.

All permanent facts created during the boot process will reside in this layer while the server remains operational. If you restart the server, whatever was stored here is gone, presumably recreated by a new call to ^CSBOOT.

Note: In a multi-bot system, you need to define this function ONCE in a common area (like a botmacro file) which is visible to all bots.

HOW DOES GC WORK HERE if you create transient facts?

You can also add facts to the boot layer from execution of user scripts. Just create facts from a user script using ^createFact(s v o #FACTBOOT) and when layer 1 is being removed at the end of the volley, these facts will migrate to the boot layer.

You may define any number of ^CSBOOT functions. This allows both multibot and standalone environments to compile without triggering a complaint about already defined functions, though only the first will ever execute.

^purgeboot (what)

If you created facts previously during user script executing (via CreateFact with a flags argument FACTBOOT) Or via fact creation in ^csboot, then they would reside there unchanged as part of a server until it goes down. Those facts are visible to all users and becaue they are system owned facts, cannot be modified by a user. However, you can use this function to remove them from the boot layer.

```
^purgeboot($variable)
```

```
^purgeboot(@1)
```

All facts you may have queried into this factset which are facts from the boot layer will be removed.

```
^purgeboot(botid)
```

All facts with the given numeric bot id in the boot layer will be removed.

If ^purgeboot is called to remove the contents of a json variable, then if that variable is not a user variable, it will be allowed to be reassigned to later, without it automatically reverting to pre-user values (which is normally what happens to changes to bot variables)

```
^CS_REBOOT()
```

```
outputmacro: ^CS_REBOOT()
```

This function, if defined by you, will be executed on every volley prior to loading user data. It is executed as a user-level program which means when it has completed all newly created facts and variables just disappear. Typically you

would have the ^CS_REBOOT function test some condition (results of a version stamp) and if the version currently loaded in the boot layer is current, it simply returns having done nothing. If the boot layer is not current, then you call the cs function ^REBOOT() which erases the current boot data and treats the remainder of the script as refilling the boot layer with new facts and variables.

Command Line Parameters

bootcmd=xxx

runs this command string before CSBOOT is run. Use it to trace the boot process or whatever.

recordboot

Will append to the toplevel file bootfacts.txt file any data moved from user layer to boot layer due to FACTBOOT facts being created. Normally facts moved to the boot layer from user layer would just be lost when the server restarts. This allows you to recover them, using ^importfact to read back these facts on server startup.