ChatScript Multiple Bots Manual

Revision 6/13/2022 cs13.2

The system can support multiple bots cohabiting the same engine. You can restrict topics to be available only to certain bots. You can also restrict facts and functions, such that variants exist per bot.

Restriction by Rule

The simplest thing is to restrict rules to being available only to certain bots by using something like

```
?: (%bot=harry ...)

?: (%bot!=harry ...).

t: (%bot=harry) My name is harry.
```

You specify which bot you want when you login, by appending `:botname` to your login name, e.g., `bruce:harry`. The demo system has only one bot, Harry. If it had two bots in it,Harry and Georgia, you could get Georgia by logging in as `yourname:georgia`. And you can confirm who she is by asking `what is your name`.

When you don't give a botname, you get the default bot. How does the system know what that is? It looks for a fact in the database whose subject will be the default bot name and whose verb is defaultbot. If none is found, the default bot is called anonymous, and probably nothing works at all. Defining the default bot is what a table does when you compile `simplecontrol.top`. It has:

```
table: ^defaultbot (^name)
^createfact(^name defaultbot defaultbot)
DATA:
harry
```

Typically when you build a level 1 topic base (e.g., `:build ben` or `:build 1` or whatever), that layer has the initialization function (`bot definition script`) for your bot(s) otherwise your bot cannot work.

This function is invoked when a new user tries to speak to the bot and tells things like what topic to start the bot in and what code processes the users input. You need one of these functions for each bot, though the functions might be pure duplicates (or might not be). In the case of Harry, the function is

```
^outputmacro: harry()
    ^addtopic(~introductions)
    $cs_control_pre = ~control
```

```
$cs_control_main = ~control
$cs_control_post = ~control
```

You can change default bots merely by requesting different build files that name the default, or by editing your table.

MULTIPLE BOTS

To have multiple bots available, one might start by making multiple folder copies of Harry and changing them to separate bots with separate `filesxxx.txt`. That's fine for building them as each stand-alone bots, but if you want them to cohabit you must have a single `filesxxx.txt` file that names the separate bot's folders- they must be built together.

If all you do is create multiple `bot definition scripts`, then the bots all equally share all outputmacro definitions, all topics, and all facts created at build time. However, facts created during conversation with a user will all be unique. A conversation between a specific bot and user results in a separate USERS topic file, so what one bot learns the others do not.

If you try to create two bots by having separate folders for each, and merely clone copies of an original folder, like `~control` and `~introductions` and `~childhood` and you will get an error message when compiling. Likely you would want only one copy of `~control` that both bots used. And either they should have different topic names for `~introductions` and `~childhood` OR you must put a bot restriction on things. See below

You can name multiple bots separated by commas with no extra spaces. E.g. `topic: ~mytopic bot=harry,roman [mykeyword]`

Restriction by Topic

A topic can be restricted to specific bots. You do this at topic definition time by naming the one or more bots (separated by commas but no spaces) allowed to use the topic.

```
topic: ~mytopic bot=Harry,Georgia REPEAT (keyword)
```

Now you learn that you can create multiple copies of the same topic, so different bots can have different copies. These form a topic family with the same name. The rules are:

- the topics share the union of keywords
- `:verify` can only verify the first copy of a topic it is allowed access to

You may create multiple copies of a topic name, which vary in their bot restrictions and content. The set of keywords given for a topic is the union of the keywords of all copies of that topic name. You should not name a topic ending in a period and a number (that is used internally to represent multiple topics of the same name).

When the system is trying to access a topic, it will check the bot restrictions. If a topic fails, it will try the next duplicate in succession until it finds a copy that the bot is allowed to use.

This means if topic 1 is restricted to Ben and topic2 is unrestricted, Ben will use topic 1 and all other bots will use topic2. If the order of declaration is flipped, then all bots including Ben will use topic 2 (which now precedes topic 1 in declaration).

You can also use the `:disable` and `:enable` commands to turn off all or some topics for a personality.

Furthermore, you don't have to use a bot restriction per topic. You can put one in a file and all topics compiled thereafter inherit that (but it is overridden if a topic has a specific value).

```
topic: ~topic1 ()
...
bot: Ben
topic: ~topic2 ()   This topic is restricted to Ben
...
```

Furthermore, you don't have to do it per file. You can do it in the filesxxx.txt build file. But you need to compile any bot definitions without a bot restriction in effect. E.g.

```
RAWDATA/BOTDEFINTIONS
bot: Harry
RAWDATA/HARRY/
bot: Georgia
RAWDATA/HARRY/
bot: 0
RAWDATA/QUIBBLE/
```

Note how we have reused all topics from Harry unchanged. That was pointless. But you could have cloned the HARRY folder for Georgia and then made modifications and then compiled that. The duplicate topics can coexist. The `bot: 0` turns off bot restrictions.

If you have bot restrictions within a file, they temporarily win over the build file one, which resumes effect when the file or topic ends.

Shared Facts

`Share` - Normally, if you have multiple bots, they all talk independently to the user. What one learns or says is not available to the other bots. It is possible to create a collection of bots that all can hear what has been said and share information. Share on a topic means that all bots will have common access/modification of a topic's state. So if one bot uses up a gambit, the other bots will not try to use that gambit themselves.

```
topic: ~mytopic SHARE REPEAT ()
```

All facts created will be visible to all bots. And if you create a permanent user variable with the starting name `$share_`, then all bots can see and modify it. So `$share_name` becomes a common variable. When sharing is in effect, the state with the user (what he said, what bot said, what turn of the volley this is, where the rejoinder mark is) is all common among the bots- they are advancing a joint conversation.

Fully Independent Bots - Restriction of Facts and Functions

Normally all facts generated during compilation are available to all bots. But you can create facts restricted to specific bots. You can have up to 64 different bots controlling different facts. You do this by setting cs_botid to a value prior to creating facts. And you output 'cs_botid = some_number in your bot definition script. The numbers form a bit mask that authorizes ownership and access to facts. For example in the bot definition script for Harry you might do $cs_botid = 1$ and for Georgia you do 'cs_botid = 2'. Each value represents a bit and should therefore be a power of 2.

When you want to create facts in tables, somewhere at the start of one bot's facts to be created, you can
set the `$cs_botid` from a table. More convenient is to use the `bot:` command to do it in either a file or a build area.

```
bot: 1 Harry
```

When you use a bot: command as a line in your `filesxxx.txt` build file, it sticks forever until some other bot: command is hit. If that is a local file command, then that affects the file but then the compile reverbs to no bot: value until the next one is hit (either globally or in a file).

This sets both the botid and the bot name, which controls what restrictions facts get (botid) and what restrictions topics get (bot name). The botid goes beyond controlling facts. It also controls ownership of outputmacro definitions. You can define different copies of functions with the same name, different arguments, different code, by making the botid be different.

You can change to a bot owner without naming any bots, in which case topics created will be usuable by any bot but facts and functions will be restricted by bot owner.

```
bot: 2
```

You disable ownership rules with

```
bot: 0
```

In the build file, you need to compile all bot definition macros and the default bot macro under bot: 0 ownership. All other bots can be built independently

by using bot ids that are powers of two and unique bot names. "‘ bot: 0 private/Mine/CONTROL/botmacro.top private/Mine1/CONTROL/botmacro.top private/Mine2/CONTROL/botmacro.top bot: 1 John private/John/ bot: 2 Harry private/Harry/ bot: 4 Martha private/Martha/

Changing bot id on the fly

^changebot(botname botid) allows a bot to pretend to be another bot and access its data, functions, and topics. Variables are not affected by this. The user topic file will remain as the user came into the server.