# ChatScript Advanced Topic Manual


Revision 8/18/2019 cs9.62

# ADVANCED TOPICS

There are several things to know about advanced topics.

## Topic Execution

When a topic is executing rules, it does not stop just because a rule matches. It will keep executing rules until some rule generates ouput for the user or something issues an appropriate `^end` or `^fail` call. So you can do things like this:

```
u: ( I love ) $userloves = true

u: ( dog ) $animal = dog

u: ( love ) Glad to hear it

u: ( dog ) I hate dogs
```

and given *I love dogs*, the system will set $userloves and $animal and output *glad to hear it*.

## Topic Control Flags

The first are topic flags, that control a topic's overall behavior. These are placed between the topic name and the (keywords list). You may have multiple flags. E.g.

```
topic: ~rust keep random [rust iron oxide]
```

The flags and their meanings are:

| flag | description |
|---|---|
| `random` | search rules randomly instead of linearly |
| `norandom` | (default) search rules linearly |

| flag | description |
|---|---|
| `keep` | do not erase responders ever. Gambits (and rejoinders) are not affected by this |
| `erase` | (default) erase responders that successfully generate output.Gambits automatically erase unless you suppress them specifically. |
| `nostay` | do not consider this a topic to remain in, leave it (except for rejoinders) |
| `stay` | (default) make this a pending topic when it generates output |
| `repeat` | allow rules to generate output which has been output recently |
| `norepeat` | (default) do not generate output if it matches output made recently |
| `priority` | raise the priority of this topic when matching keywords |
| `normal` | (default) give this topic normal priority when matching keywords |

| flag | description |
|---|---|
| `deprioritize` | lower the priority of this topic when matching keywords |
| `system` | this is a system topic. It is automatically `nostay`, `keep`.`keep` automatically applies to gambits as well. The system never looks to these topics for gambits. System topics can never be considered pending (defined shortly). They can not have themselves or their rules be enabled or disabled. Their status/data is never saved to user files. |
| `user` | (default) this is a normal topic |
| `noblocking` | should not perform any blocking tests on this topic in `:verify` |
| `nopatterns` | should not perform any pattern tests on this topic in `:verify` |

| flag | description |
|---|---|
| `nosamples` | should not perform any sample tests on this topic in `:verify` |
| `nokeys` | should not perform any keyword tests on this topic in `:verify` |
| `more` | normally if you try to redeclare a concept, you get an error. `more` tells CS you intend to extend the concept and allows additional keywords. |
| `bot=name` | if this is given, only named bots are allowed to use this topic. See ChatScript Multiple Bots manual. |

## Rules that erase and repeat

Normally a rule that successfully generates output directly erases itself so it won't run again. Gambits do this and responders do this.

Gambits will erase themselves even if they don't generate output. They are intended to tell a story or progress some action, and so do their thing and then disappear automatically.

Rejoinders don't erase individually, they disappear when the rule they are controlled by disappears. A rule that is marked keep will not erase itself. Nor will responders in a topic marked keep (but gambits still will).

Responders that generate output erase themselves. Responders that cause others

to generate output will not normally erase themselves (unless. . . ):

```
u: ( * ) respond(~reactor)
```

If the above rule causes output to be generated, this rule won't erase itself, the rule invoked from the `~reactor` topic that actually generated the output will erase itself. But, if the rule generating the output is marked keep, then since someone has to pay the price for output, it will be this calling rule instead.

`Repeat` does not stop a rule from firing, it merely suppresses its output. So the rule fires, does any other effects it might have, but does not generate output. For a responder, if it doesn't generate output, then it won't erase itself. For a gambit, it will because gambits erase themselves regardless of whether they generate output or not.

## Keywords vs Control Script

A topic can be invoked as a result of its keywords or by a direct call from the control script or some other topic. If you intend to call it from script, then there is almost never any reason to give it keywords as well, because that may result in it being called twice, which is wasteful, or out of order, if there was a reason for the point you called it from script.

## Pending Topics

The second thing to know about topics is what makes a topic pending. Control flow passes through various topics, some of which become pending, meaning one wants to continue in those topics when talking to the user. Topics that can never be pending are: system topics, blocked topics (you can block a topic so it won't execute), and nostay topics.

What makes a remaining topic pending is one of two things. Either the system is currently executing rules in the topic or the system previously generated a user response from the topic. When the system leaves a topic that didn't say anything to the user, it is no longer pending. But once a topic has said something, the system expects to continue in that topic or resume that topic.

The system has an ordered list of pending topics. The order is:

- 1st- being within that topic executing rules now,
- 2nd- the most recently added topic (or revived topic) is the most pending.

You can get the name of the current most pending topic `(%topic)`, add pending topics yourself `(^addtopic())`, and remove a topic off the list `(^poptopic())`. Issuing ^poptopic(x) or ^fail(TOPIC x) will remove it from the pending list. Entering a topic already deep within the pending list will move it to the front.

You can request ^gambit(pending) or ^respond(pending) to have CS scan that list for a response. The system does not automatically USE the pending topic list to control your flow. That is for you to manage. It merely tracks it for you.

## Random Gambit

The third thing about topics is that they introduce another type, the random gambit, `r:`.

The topic gambit `t:` executes in sequence forming in effect one big story for the duration of the topic. You can force them to be dished randomly by setting the random flag on the topic, but that will also randomize the responders. And sometimes what you want is semirandomness in gambits.

That is, a topic treated as a collection of subtopics for gambit purposes. This is `r:` The engine selects an `r:` gambit randomly, but any `t:` topic gambits that follow it up until the next random gambit are considered "attached" to it. They will be executed in sequence until they are used up, after which the next random gambit is selected.

```
Topic: ~beach [beach sand ocean sand_castle]

# subtopic about swimming
r: Do you like the ocean?

t: I like swimming in the ocean.

t: I often go to the beach to swim.

# subtopic about sand castles.
r: Have you made sand castles?
    a: (~yes) Maybe sometime you can make some that I can go see.
    a: (~no) I admire those who make luxury sand castles.

t: I've seen pictures of some really grand sand castles.
```

This topic has a subtopic on swimming and one on sand castles. It will select the subtopic randomly, then over time exhaust it before moving onto the other subtopic.

Note any `t:` gambits occurring before the first `r:` gambit, will get executed linearly until the `r:` gambits can fire.

## Overview of the control script

Normally you start using the system with the pre-given control script. But it's just a topic and you can modify it or write your own.

The typical flow of control is for the control script to try to invoke a pending rejoinder. This allows the system to directly test rules related to its last output, rules that anticipate how the user will respond.

Unlike responders and gambits, the engine will keep trying rejoinders below a rule until the pattern of one matches and the output doesn't fail.

Not failing does not require that it generate user output. Merely that it doesn't return a fail code. Whereas responders and gambits are tried until user output is generated (or you run out of them in a topic).

If no output is generated from rejoinders, the system would test responders. First in the current topic, to see if the current topic can be continued directly. If that fails to generate output, the system would check other topics whose keywords match the input to see if they have responders that match. If that fails, the system would call topics explicitly named which do not involve keywords. These are generic topics you might have set up.

If finding a responder fails, the system would try to issue a gambit. First, from a topic with matching keywords. If that fails, the system would try to issue a gambit from the current topic. If that fails, the system would generate a random gambit.

Once you find an output, the work of the system is nominally done. It records what rule generated the output, so it can see rejoinders attached to it on next input. And it records the current topic, so that will be biased for responding to the next input. And then the system is done. The next input starts the process of trying to find appropriate rules anew.

There are actually three control scripts (or one invoked multiple ways). The first is the preprocess, called before any user sentences are analyzed. The main script is invoked for each input sentence. The postprocess is invoked after all user input is complete. It allows you to examine what was generated (but not to generate new output except using special routines `^postprintbefore` and `^postprintafter`).