# Beginners Guide to CEML

Thanks for taking an interest in CEML, the Coordinated Event Modeling Language. Using CEML you'll be able to get people to do whatever you want, whether it's evacuating a submarine, finding a gym buddy, helping people nearby, or just throwing a great party.

A CEML *script* is a recipe for action. Just like a recipe, it has three parts: a title, a list of ingredients, and then a section that says what to do with the ingredients. Here's an example:

```
"Trade favorite colors"
gather 1 guy and 1 girl within 50 feet
ask guy re favorite_color:
    What's your favorite color?
tell girl:
    Someone nearby likes the color |guy.favorite_color|.  Find them.
```

In the example, the first line is the title, which is written in double quotes. The second line is the ingredients and how they are obtained. In CEML, the ingredients are usually people (although sometimes they can be places, things, or notes). The remaining lines are the instructions.

A good way to learn how flexible CEML is is to write the same script a different way. Then you can see which parts can change and which parts have to stay the same:

```
"Trade favorite colors"
gather a, b within 50ft
ask a re x: What's your favorite color?
tell b: Someone nearby likes the color |a.x|.  Find them.
```

Whether you write a script compactly with short names for roles and answers or long names, it runs exactly the same.

## Commands

The above script only uses three CEML commands–*gather*, *ask*, and *tell*–but there aren't many more commands to learn. Only six more, actually: besides *gather*, the other ingredients commands are *await* and *nab*. Besides *ask* and *tell*, the other instructions commands are *assign*, *release*, *expect*, and *sync*.

As a sneak preview, here's an example that uses the other commands:

```
"Emergency Medicine"
await 1 concerned patient
```

```
nab a doctor within 5 miles
ask patient re problem: What's wrong?
assign doctor to patient: Attend to patient's |problem|.
release doctor as responsive
```

## Texts

Commands like *ask*, *tell*, and *assign* contain texts which are delivered to the player who is the subject of the command. These texts appears after a colon, and can appear on the same or the following lines. If they appears on the following lines, they must be indented, as in the very first example above or the one following.

```
"Signups"
await 1 new signup
tell signup:
    We are so glad
    that you have
    signed up.
```

Texts may also contain hyperlinks, which if the player has a smartphone with a web browser, will be openable.

```
"Hostility"
gather enemy, reporter within 1 block
ask reporter re clothing: What are you wearing?
assign enemy: Find someone wearing |clothing| and harass them.
assign reporter:
    Someone will harass you.  Notice how it feels.
    Then fill out this google spreadsheet form
    to describe your experience.
    http://spreadsheet.google.com/foo
tell both: thanks so much for participating!
```

## Roles

In the examples so far, the words like ‘enemy’, ‘reporter’, ‘guy’, ‘girl’, ‘a’, ‘b’, ‘signup’, ‘patient’, and ‘doctor’ are *role names*. Most of the time, a role name can be any word you like. They are just a placeholder to connect *casting commands* like *gather* and *await*, with *coordination commands* like *tell*.

Squads on Groundcrew, however, can define special meanings for certain roles. So on a particular squad, a ‘doctor’ might mean someone who’s been tagged/released with the tag ‘doctor’, and a patient might mean anyone else.

Some role names are always special. For instance, if you use the role name ‘organizer’, it always means someone who’s an official organizer for that squad. And if you use the role names ‘both’, ‘all’, ‘each’, or ‘everyone’, it means that the instruction applies to everyone regardless of how they were *cast*.

```
"Get Help"
await concerned user
nab organizer
tell organizer: someone's not doing well.
```

When there is only one kind of role in a script, it is possible to omit the role name in instructional commands.

```
await 1 tired user
ask re cause: why are you tired?
tell: i hope you feel better
```

## Social Information

Texts (see above) may contain *social information* that's inserted from answers by another player. To do this, we use a section surrounded by vertical bars (|). In the text that's sent to the player, this section will be replaced by the other player's answer.

```
"Empire"
gather emperor and 2-5 servants
ask emperor re tasks: What should your servants do today?
assign servants: The emperor has asked you to do |tasks|. Do them now.
```

In situations where there may be confusion about which answer you mean, you can specify a particular role by using a dot (.) in between the role and the answer names.

```
"Cheese Monte"
gather a,b,c
ask each re cheese: What kind of cheese do you have?
tell a: Someone near you has |c.cheese|. Take it!
tell b: Someone near you has |a.cheese|. Take it!
tell c: Someone near you has |b.cheese|. Take it!
```

## Assign

It is probably clear what *ask* and *tell* do, but we haven't exactly covered *assign*. *Assign* gives a task which is expected to last a certain duration. While a *tell* command for a particular player completes immediately and goes on to the next command, an *assign* will wait for the player to say they've completed the task, and collect photo and textual reports as they do it. There are different representations for this depending on whether the player is connected to the script via text messaging, the iphone, or the mobile web.

A special form of *assign* will additionally direct the player to a person or place where they are supposed to go.

```
"Streetcorner Observation"
```

```
gather player, streetcorner within 5 blocks
assign player to streetcorner:
    Go there and report what you observe.
```

On iPhone or mobile web, the player will get a little map to direct them. There are several special *role names* which are used to indicate places rather than people. These include “streetcorner”, “park”, “field”, and “landmark”.

## Expect

When you ask a question using *ask*, you may not want to let a player proceed unless they’ve answered in a way you understand or recognize. This can be accomplished using *expect*:

```
ask player re path:
    Do you prefer to the path to the left or the right?
expect /left|right/ from player:
    Please say "left" or "right"!
```

The expect statement, like the release statement described below, can be passed strings, certain keywords, or regular expressions.

## Sync up

There also may be times when you want to sync up several players and make sure they have both completed assignments or answered questions before they each proceed with the script:

```
assign redcoat:
    March briskly towards Concord
assign minuteman:
    Hide in the bushes
sync up redcoat and minuteman
tell both:
    Prepare to be surprised.
```

## Choosing Players: Await, Gather, and Nab.

So what is really the difference between *await*, *gather*, and *nab*, you may be asking? Or perhaps you have already figured it out.

*Await* defines a kind of trigger–as soon as the conditions awaited for are met, the script will run.

```
await 2-5 level=1 players within 50ft
assign: shout out "woo-hoo!"
release as level=2
```

Unless there's an await statement in your script, it will have to be executed manually by an organizer.

*Gather* issues invitations. It will keep issuing invitations and processing people's replies to them until it has the requisite number of players for its roles.

```
"Pick-up basketball"
gather 4-8 basketball players within 4 blocks
```

*Nab* just involves people, straight up, without asking their permission or issuing them an invitation.

By default, the people that triggered an *await* script are nabbed, not gathered. That means they will never receive an invitation. If you want to make sure they accept, you need an *await* line and a *gather* line for the same role name.

```
"A study about vomitting"
await 5 sick users
gather 2-5 users
ask re feeling: how do you feel?
```

## Qualifying players

You may have noticed that some of the *await*, *gather*, and *nab* statements have adjectives or qualifiers before the role name. For instance, the word "sick" in "sick users" above is such a qualifier. So is "basketball", "level1", "tired", "concerned", and "new".

Some of these have special meanings: the qualifier *new* is automatically applied to new signups. So if your script has an *await* command that looks for someone new, it will automatically run on signup.

```
await 3 new signups
assign:
    You have all just signed up!
    Take a photo of something you
    love and share with one another.
```

Another qualifier with a special meaning is *concerned*. This is applied to players who, because of their text messaging or their tweeting or their interaction with the iphone or web apps, seem like are confused or have an urgent question or issue.

The other qualifiers don't mean anythings special, but they select only players who have been released or tagged with that particular word. So "gather 5-20 level=1 users" will only invite users who have been tagged or released as level=1.

## Finer control of matching

The *await* keyword supports a variety of options that let you match players with more specificity. For instance, you may wish to only match players that text in within 10 minutes of one another:

```
await 2 new players over 10 minutes
```

Or players that have the same favorite color:

```
await 2 new players with matching favorite_color
```

Here's two scripts that cooperate to put people in groups by favorite color:

```
await new signup
ask signup re name: What's your name?
ask signup re favorite_color: What's your favorite color?

await 2 players with matching favorite_color
tell players: |buddy.name| also likes |favorite_color|.
```

## Releasing

When a player has successfully completed their role in your script, you can add or change the tags associated with that player using the *release* command.

```
await 3 new signups
assign:
    You have all just signed up!
    Take a photo of something you
    love and share with one another.
release as photo_sharing
```

You can also release a player early, subject to some conditions, by using *if* or *unless*:

```
await 1 boss and 1 worker
ask worker:
    Do you feel good about working today?
release worker unless yes
ask worker re skill:
    What are you good at?
ask boss re job:
    You have a worker with skill |skill|.  What should they do?
```

There are some special keywords you can use after if or unless. These include "yes", "no", "done", "okay". You can also use a string in quotes ("activate"), in which case their answer is compared to the string case insensitively and with whitespace trimmed. Or you can use a regular expression surrounded by forward slashes (/^red|blue|green$/) for more exact matching.

## Making Connections between Scripts

*Release* and qualifications can be used to link scripts together. The release at the end of the first script can trigger the *await* statement in the next script. Here’s an example:

```
await 20 stage=photo_sharing users
ask re opinion:
    How did you like sharing photos just now?
release as stage=signed_up
```

## Congratulations

You now know the basics of CEML and can start brainstorming your own scripts. If you don’t have a squad already, contact us at Groundcrew and we’ll get you set up.

We will also be creating a google group for CEML developers soon. Write info@groundcrew.us to make sure you’re on it.

---

## FAQ

### Do I need a title for every script?

You may have noticed that some of the script examples given in this guide do not have titles. Titles can be omitted under certain conditions. The main issue is that if a script has a *gather* statement then it MUST have a title. This is because the invitations that are issued contain the script title. Otherwise, titles are still often good to have: if the script is run manually it must have a title so that organizers can select it, and when users are running one of our smartphone clients like the iPhone app or the HTML5 webpage, they will see the title if there is one. The title will also be associated with any long-lasting media and reports that come from running the script.

### What if I want to share documents or notes in a script?

The February or March 2011 version of CEML will support passing and curating notes and documents between scripts. This will allow for a mix of flashmob and knowledge work-style coordination. Syntax is not finalized, but something like:

```
await designer
assign:
    sketch out a design for something you
    want to build and take a photo
document as designer_photo
```

```
release designer_photo as unreviewed

await 6 reviewers and unreviewed designer_photo document
ask reviewers re opinion:
    What do you think of this idea? |designer_photo|
release designer_photo as reviewed not unreviewed
```