

Web page syntax

Basics

Basic webpages

To create a webpage for a Dynet server, simply make a file in the site directory on the server. Edit this and add some text, for example “Hello, world!”

To access this page, go to a client and type in the website’s name, followed by a forward slash, followed by the name of the file. If you called it “hello”, then type “example/hello” into the address bar to see your web page.

Special webpages

There are 2 special webpages in the “site” folder, and a third file that isn’t an actual webpage, called “dynamic”. Ignore the dynamic one for now, and let’s look at the special pages:

home: The page named “home” is the page the user is redirected to if he does not specify a page. If he goes to the website called “example” by simply typing in “example”, he will be redirected to “example/home”, or this page. It’s very important and should have a brief introduction, important information and/or links to the most important pages

404: if a page is not found, the server will load this page instead. In it, you should apologise to the user (even if it’s not your fault) and provide a link to the home page or something.

Adding colours

So now you have some text, but you don’t like the default black and white colours. Don’t worry, Dynet supports 16 background and foreground colours.

To change the colour of text, type the following before it: “^fX” where X is any colour number or letter from the list below. To make your text pink, use “^f6”, for example.

To change the background colour (the colour behind the text), use “^bX”, where X is your colour like before.

So, to recap, “^f” is for foreground colours,

And “^b” is for background colours.

Color name	Color number
white	0
orange	1
magenta	2
Light blue	3

Yellow	4
Lime	5
Pink	6
Gray	7
Light gray	8
Cyan	9
Purple	A
Blue	B
Brown	C
Green	D
Red	E
Black	F

Making web pages interactive

So now you have a page with pretty colours and text. Now, if you've ever used the internet before, you would know that there's a lot of stuff you can click on that will take you to different web pages or make other things happen. Let's add that now!

Dynet's link system is based on lines. Each horizontal line can contain one link. You see, each line is made up of 2 parts: the visible part and the invisible part. So far, you've been making text and colours. This is all in the visible part.

What separates the two is the ~ character (top left of your keyboard usually, press shift to type it). Everything to the left of the ~ is visible, and everything to the right of it is not, and will only do something when that line is selected. If you don't use the ~ character in a line, clicking it will do nothing.

So what can we type in the invisible part? Here's the full list:

Function	What clicking it does	Variables/notes
~loc:page	Brings the user to a different page on this server	Replace "page" with the name of the webpage (for example "home")
~glob:site/page	Brings the user to another website	Replace site with the website name and page with the page's name on that website
~ref	Refreshes the page (dynamic lines get updated)	This will be useful when using dynamic lines
~ask:cookie:question	Shows the user a dialog box asking for input	This will be described in further detail later
~\$custom	Runs custom lua code, so clicking it can do anything	Again, further detail later

So yes, for now you only need to worry about the first 2 ones. Try making multiple web pages and linking them to each other (or don't, I don't mind)

Intermediate

Warning: This section will require you to program in Lua and be familiar with the computercraft APIs. I won't go into detail on how these APIs work or how to use them, or how to code in Lua; I will assume you already know this.

You will also need to have fully read the basic section.

The dynamic file

As previously mentioned, there's a special file in the site folder- the dynamic file. Anything dynamically generated will be written in here, and any other lua code you may want to write.

By default, there's 3 commands defined:

```
getDynamicLine()
```

```
runDynamicCommand()
```

```
takeDynamicInput()
```

These functions are executed by the Dynet server program when dynamic data is received or needed. Anything written outside these functions will be executed on startup, before the server is started. Any setup functions or variables you need to declare or files you need to load can be added here.

getDynamicLine: Making dynamic content

Dynet stands for "Dynamic internet", so it's designed for dynamic content. When the visible part of a line in a web page starts with a \$, the text following the \$ character is the dynamic line's name, up until the ~ character. When the server is sending a web page and it encounters a dynamic line, it will call this function and tell it the dynamic line name (e.g. "date" for \$date) and what page this dynamic line was found on.

Most of the time you will only need to worry about the "name" argument, and knowing what page this happened on is for more advanced uses. The value this function returns will be displayed instead of the dynamic line name.

This is a little hard to understand, so here's an example. Replace the getDynamicLine command with the following code:

```
visitCount=0
function getDynamicLine(name,page)
    if name=="visitors" then
        visitCount=visitCount+1
        return "this site has been visited "..visitCount.." times"
    end
    return "dynamic line not found"
end
```

and add “\$visitors” as a line on your web page.

Every time you visit or refresh the page, the visitor count will go up.

Some things to try:

- This value will reset when the website is restarted. Why not store its value in a file?
- You can also make it count the visitors for individual pages, using the “page” argument
- Try using the colour commands in the output as well, they will still work!

This is just a simple example, but much more can be done with this. It’s all up to you! And remember, colour commands work just fine if they are returned!

runDynamicCommand: Make links do cool stuff

Making links redirect the user to pages is boring. What if you could make buttons that did something cooler?

Dynamic commands can be used to execute lua code, too! You can output redstone signals, move around (if the server is a wireless turtle), play music from disks, anything!

This function gives 3 arguments: name, client, and page. “client” contains the ID of the client who clicked the link. Coding works in exactly the same way as getDynamicLine, except there’s no return value. Any computercraft/lua API can be used here.

If you want the client to react in some way when the link is clicked, you can send instructions back to the client with the rednet API. For example, to refresh the page (to, for example, refresh values the function has updated), use `rednet.send(client,“ref”)`. All functions can be used through this (see the table from the “making webpages interactive” section), just remember to exclude the ~ character when sending them.

The best way to learn how this works is to experiment and look at the example pages.

Advanced

Alright, so you have dynamic content on your web pages, but you want more POWER. How about a fully dynamic web page with fully dynamic links generated entirely in Lua? This is possible! Do note, however, this is only for some really fancy dynamic content like auto-generated pages.

Unfortunately, this section of the manual is still under construction. I’ll add this info later, sorry!