



Custom Components

The majority of the time the default [ft components](#) are all you need (for example `Div`, `P`, `H1`, etc.).

Pre-requisite Knowledge

If you don't know what an ft component is, you should read [the explaining ft components explainer first](#).

However, there are many situations where you need a custom ft component that creates a unique HTML tag (for example `<zero-md></zero-md>`). There are many options in FastHTML to do this, and this section will walk through them. Generally you want to use the highest level option that fits your needs.

Real-world example

[This external tutorial](#) walks through a practical situation where you may want to create a custom HTML tag using a custom ft component. Seeing a real-world example is a good way to understand why the contents of this guide is useful.

NotStr

The first way is to use the `NotStr` class to use an HTML tag as a string. It works as a one-off but quickly becomes harder to work with as complexity grows. However we can see that you can generate the same xml using `NotStr` as the out-of-the-box components.

```
from fasthtml.common import NotStr, Div, to_xml
```

```
div_NotStr = NotStr('<div></div>')
print(div_NotStr)
```

```
<div></div>
```

Automatic Creation

The next (and better) approach is to let FastHTML generate the component function for you. As you can see in our `assert` this creates a function that creates the HTML just as we wanted. This works even though there is not a `Some_never_before_used_tag` function in the `fasthtml.components` source code (you can verify this yourself by looking at the source code).

Tip

Typically these tags are needed because a CSS or Javascript library created a new XML tag that isn't default HTML. For example the `zero-md` javascript library looks for a `<zero-md></zero-md>` tag to know what to run its javascript code

on. Most CSS libraries work by creating styling based on the `class` attribute, but they can also apply styling to an arbitrary HTML tag that they made up.

```
from fasthtml.components import Some_never_before_used_tag
```

```
Some_never_before_used_tag()
```

```
<some-never-before-used-tag></some-never-before-used-tag>
```

Manual Creation

The automatic creation isn't magic. It's just calling a python function `__getattr__` and you can call it yourself to get the same result.

```
import fasthtml

auto_called = fasthtml.components.Some_never_before_used_tag()
manual_called = fasthtml.components.__getattr__('Some_never_before_used_tag')()

# Proving they generate the same xml
assert to_xml(auto_called) == to_xml(manual_called)
```

Knowing that, we know that it's possible to create a different function that has different behavior than FastHTMLs default behavior by modifying how the `__getattr__` function creates the components! It's only a few lines of code and reading that what it does is a great way to understand components more deeply.

Tip

Dunder methods and functions are special functions that have double underscores at the beginning and end of their name. They are called at specific times in python so you can use them to cause customized behavior that makes sense for your specific use case. They can appear magical if you don't know how python works, but they are extremely commonly used to modify python's default behavior (`__init__` is probably the most common one).

In a module `__getattr__` is called to get an attribute. In `fasthtml.components`, this is defined to create components automatically for you.

For example if you want a component that creates `<path></path>` that doesn't conflict names with `pathlib.Path` you can do that. FastHTML automatically creates new components with a 1:1 mapping and a consistent name, which is almost always what you want. But in some cases you may want to customize that and you can use the `ft_hx` function to do that differently than the default.

```
from fasthtml.common import ft_hx

def ft_path(*c, target_id=None, **kwargs):
    return ft_hx('path', *c, target_id=target_id, **kwargs)

ft_path()
```

```
<path></path>
```

We can add any behavior in that function that we need to, so let's go through some progressively complex examples that you may need in some of your projects.

Underscores in tags

Now that we understand how FastHTML generates components, we can create our own in all kinds of ways. For example, maybe we need a weird HTML tag that uses underscores. FastHTML replaces `_` with `-` in tags because underscores in tags are highly unusual and rarely what you want, though it does come up rarely.

```
def tag_with_underscores(*c, target_id=None, **kwargs):
    return ft_hx('tag_with_underscores', *c, target_id=target_id, **kwargs)

tag_with_underscores()
```

```
<tag_with_underscores></tag_with_underscores>
```

Symbols (ie @) in tags

Sometimes you may need to use a tag that uses characters that are not allowed in function names in python (again, very unusual).

```
def tag_with_AtSymbol(*c, target_id=None, **kwargs):
    return ft_hx('tag-with-@symbol', *c, target_id=target_id, **kwargs)

tag_with_AtSymbol()
```


```
<tag-with-@symbol></tag-with-@symbol>
```

Symbols (ie @) in tag attributes

It also may be that an argument in an HTML tag uses characters that can't be used in python arguments. To handle these you can define those args using a dictionary.

```
Div(normal_arg='normal stuff',**{'notNormal:arg:with_varing@symbols!':'123'})
```

```
<div normal-arg="normal stuff" notnormal:arg:with_varing@symbols!="123"></div>
```

 Report an issue