Advanced features

The features listed on this page fall outside the rest of the main categories.

`#[cfg(doc)]`: Documenting platform-specific or feature-specific information

For conditional compilation, Rustdoc treats your crate the same way the compiler does. Only things from the host target are available (or from the given `--target` if present), and everything else is "filtered out" from the crate. This can cause problems if your crate is providing different things on different targets and you want your documentation to reflect all the available items you provide.

If you want to make sure an item is seen by Rustdoc regardless of what platform it's targeting, you can apply `#[cfg(doc)]` to it. Rustdoc sets this whenever it's building documentation, so anything that uses that flag will make it into documentation it generates. To apply this to an item with other `#[cfg]` filters on it, you can write something like `#[cfg(any(windows, doc))]`. This will preserve the item either when built normally on Windows, or when being documented anywhere.

Please note that this `cfg` is not passed to doctests.

Example:

```rust
/// Token struct that can only be used on Windows.
#[cfg(any(windows, doc))]
pub struct WindowsToken;
/// Token struct that can only be used on Unix.
#[cfg(any(unix, doc))]
pub struct UnixToken;
```

Here, the respective tokens can only be used by dependent crates on their respective platforms, but they will both appear in documentation.

Interactions between platform-specific docs

Rustdoc does not have a magic way to compile documentation 'as-if' you'd run it once for each platform (such a magic wand has been called the 'holy grail of rustdoc'). Instead, it sees *all* of your code at once, the same way the Rust compiler would if you passed it `--cfg doc`. However, Rustdoc has a trick up its sleeve to handle platform-specific code if it *does* receive it.

To document your crate, Rustdoc only needs to know the public signature of your functions. In particular, it doesn't have to know how any of your functions are implemented, so it ignores all type errors and name resolution errors with function bodies. Note that this does *not* work for anything outside a function body: since Rustdoc documents your types, it has to know what those types are! For example, this code will work regardless of the platform:

```rust
pub fn f() {
    use std::os::windows::ffi::OsStrExt;
}
```

but this will not, because the unknown type is part of the function signature:

```
pub fn f() -> std::os::windows::ffi::EncodeWide<'static> {
    unimplemented!()
}
```

For a more realistic example of code this allows, see the rustdoc test suite.

Add aliases for an item in documentation search

This feature allows you to add alias(es) to an item when using the `rustdoc` search through the `doc(alias)` attribute. Example:

```
#[doc(alias = "x")]
#[doc(alias = "big")]
pub struct BigX;
```

Then, when looking for it through the `rustdoc` search, if you enter "x" or "big", search will show the `BigX` struct first.

There are some limitations on the doc alias names though: you can't use `"` or whitespace.

You can add multiple aliases at the same time by using a list:

```
#[doc(alias("x", "big"))]
pub struct BigX;
```

Custom search engines

If you find yourself often referencing online Rust docs you might enjoy using a custom search engine. This allows you to use the navigation bar directly to search a `rustdoc` website. Most browsers support this feature by letting you define a URL template containing `%s` which will be substituted for the search term. As an example, for the standard library you could use this template:

```
https://doc.rust-lang.org/stable/std/?search=%s
```

Note that this will take you to a results page listing all matches. If you want to navigate to the first result right away (which is often the best match) use the following instead:

```
https://doc.rust-lang.org/stable/std/?search=%s&go_to_first=true
```

This URL adds the `go_to_first=true` query parameter which can be appended to any `rustdoc` search URL to automatically go to the first result.