Naming conventions

General conventions [RFC #430]

The guidelines below were approved by RFC #430 (https://github.com/rust-lang/rfcs/pull/430).

In general, Rust tends to use `CamelCase` for "type-level" constructs (types and traits) and `snake_case` for "value-level" constructs. More precisely:

Item	Convention
Crates	`snake_case` (but prefer single word)
Modules	`snake_case`
Types	`CamelCase`
Traits	`CamelCase`
Enum variants	`CamelCase`
Functions	`snake_case`
Methods	`snake_case`
General constructors	`new` or `with_more_details`
Conversion constructors	`from_some_other_type`
Local variables	`snake_case`
Static variables	`SCREAMING_SNAKE_CASE`
Constant variables	`SCREAMING_SNAKE_CASE`
Type parameters	concise `CamelCase`, usually single uppercase letter: `T`
Lifetimes	short, lowercase: `'a`

In `CamelCase`, acronyms count as one word: use `Uuid` rather than `UUID`. In `snake_case`, acronyms are lower-cased: `is_xid_start`.

In `snake_case` or `SCREAMING_SNAKE_CASE`, a "word" should never consist of a single letter unless it is the last "word". So, we have `btree_map` rather than `b_tree_map`, but `PI_2` rather than `PI2`.

Referring to types in function/method names [RFC 344]

The guidelines below were approved by RFC #344 (https://github.com/rust-lang/rfcs/pull/344).

Function names often involve type names, the most common example being conversions like `as_slice`. If the type has a purely textual name (ignoring parameters), it is straightforward to convert between type conventions and function conventions:

Type name	Text in methods
`String`	`string`
`Vec<T>`	`vec`
`YourType`	`your_type`

Types that involve notation follow the convention below. There is some overlap on these rules; apply the most specific applicable rule:

Type name	Text in methods
`&str`	`str`
`&[T]`	`slice`
`&mut [T]`	`mut_slice`
`&[u8]`	`bytes`
`&T`	`ref`
`&mut T`	`mut`
`*const T`	`ptr`
`*mut T`	`mut_ptr`

Avoid redundant prefixes [RFC 356]

> The guidelines below were approved by RFC #356 (https://github.com/rust-lang/rfcs/pull/356).

Names of items within a module should not be prefixed with that module's name:

Prefer

```
mod foo {
    pub struct Error { ... }
}
```

over

```
mod foo {
    pub struct FooError { ... }
}
```

This convention avoids stuttering (like `io::IoError`). Library clients can rename on import to avoid clashes.

Getter/setter methods [RFC 344]

> The guidelines below were approved by RFC #344 (https://github.com/rust-lang/rfcs/pull/344).

Some data structures do not wish to provide direct access to their fields, but instead offer "getter" and "setter" methods for manipulating the field state (often providing checking or other functionality).

The convention for a field `foo: T` is:

- A method `foo(&self) -> &T` for getting the current value of the field.
- A method `set_foo(&self, val: T)` for setting the field. (The `val` argument here may take `&T` or some other type, depending on the context.)

Note that this convention is about getters/setters on ordinary data types, *not* on builder objects (../ownership/builders.html).

Escape hatches [FIXME]

[FIXME] Should we standardize a convention for functions that may break API guarantees? e.g. `ToCStr::to_c_str_unchecked`

Predicates

- Simple boolean predicates should be prefixed with `is_` or another short question word, e.g., `is_empty`.
- Common exceptions: `lt`, `gt`, and other established predicate names.