# THE WIT CHEATSHEET



# **File Structure**

#### Package

A single package can and often is represented by multiple files. A common pattern is types.wit, world.wit, and my-interface-name.wit. The package must include a namespace and an identifier, and can optionally include a version number which should follow semantic versioning.

#### Interface

A collection of types and functions scoped to a package which can be used within a world. Interfaces are the only place that a type can be defined. Packages can contain multiple interfaces.

#### World

Akin to a complete description of a component. A world is a collection of imports and exports that allow the component to interact with the host as well as other components.

# Language Keywords

These keywords are used to describe some sort of action or property within a component.

include	used to include worlds into another
use	will include types from an interface in a world
import	used to import an entire interface into a world
export	used to export an interface/func from a world

# **Named Types**

Types listed here all require an identifier. They provide more structure and flexibility than primitive types.

record	a collection of types accessible via named keys
enum	a type that can equal one of a set of values
variant	a type that be one of many predefined types
flags	a collection of flags that can be toggled independantly from each other
func	a function type
resource	a new abstract resource type. See below

### **Resource Keywords**

Keywords here relate to resources. Resources are used to describe variables that should not be copied by value.

constructor	method that returns a handle of the containing resource type
static	mark a resource func as scoped to resource
self	the borrowed handle to the resource
borrow	mark a handle as a temporary loan from caller to callee
owned	a handle representing unique ownership of a resource

# WASI Component Types

#### Command

A component defined as a Command has a main function (or in wasi:cli, a run function) and terminates when the function returns.

#### Reactor

This component is more like a library. It exposes an API once it is instantiated and remains live, allowing functions on the component to be called from the host or other components.

# **Primitive Types**

These are the core types for a wasm module. Most languages will implement all of these types.

bool	boolean value; true or false
s8, s16, s32, s64	signed integers with 8, 16, 32 or 64 bit length
u8, u16, u32, u64	unsigned integers with 8, 16, 32 or 64 bit length
float32, float64	floating-point numbers with 32 or 64 bit length
char	single Unicode scalar character
string	Unicode string with a finite length

# **Container Types**

These types are used to create collections of other types. They can contain any of the primitive or named types.

<pre>tuple&lt;&gt;</pre>	a finite sequence of values of different types
list <t></t>	a sequence of values of the type T
option <t></t>	mark a type as optional, value will be type T or no value
result <t,e></t,e>	represents value or error, where the output will be one or the other but not both

### Comments

Sometimes you just need to leave a note.

/// single line comment

/\* comment block \*/

# WASI Worlds to know

These are some useful worlds defined by the WASI proposals that you might find useful in writing your own components.

wasi:cli/command wasi:keyvalue/keyvalue wasi:blob-store/blob-store wasi:messaging/messaging wasi:nn/ml wasi:http/proxy wasi:cloud-core/cloud-core

# Cosmonic THE FUTURE OF DISTRIBUTED APPLICATIONS, TODAY

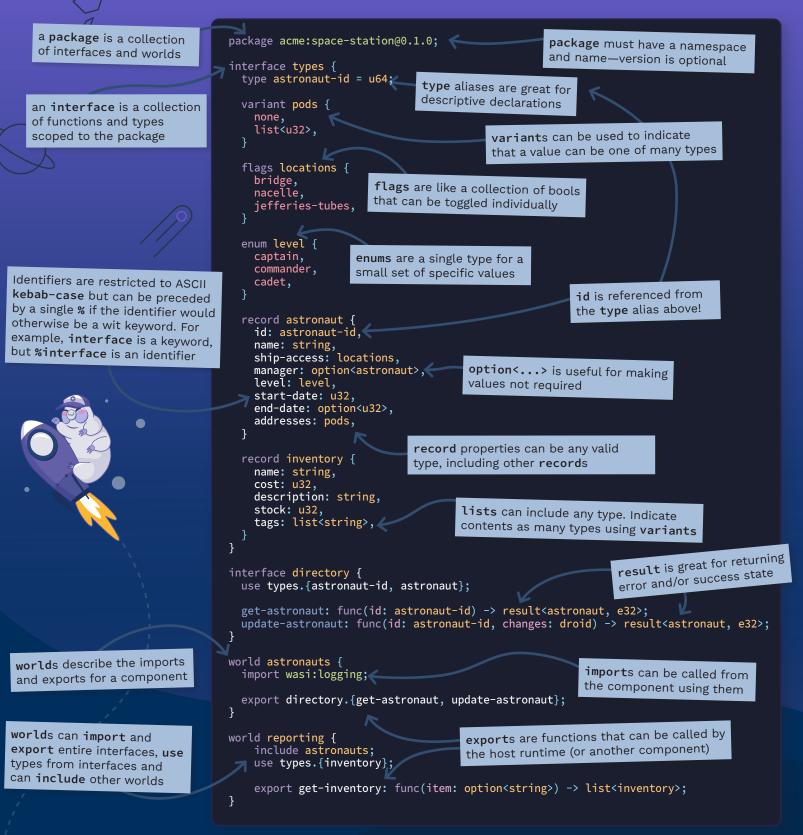
From napkin sketch to running apps anywhere, at scale, in minutes, Cosmonic is the lightweight, low-boilerplate platform that radically simplifies application development. Build your apps with composable Wasm components that run in any datacenter, cloud or edge.



wasmCloud



# WHAT DOES WIT LOOK LIKE?



# COSMONIC THE FUTURE OF DISTRIBUTED APPLICATIONS, TODAY

From napkin sketch to running apps anywhere, at scale, in minutes, Cosmonic is the lightweight, low-boilerplate platform that radically simplifies application development. Build your apps with composable Wasm components that run in any datacenter, cloud or edge. Cosmonic





