

PHANTASM: The Abstract Grammar Cheatsheet

This cheatsheet summarizes the PHANTASM abstract grammar in loose BNF-style pseudocode. The token grammar is not expanded. Instead, the following names are used to represent the various token types:

- **Number**: A number literal token.
- **String**: A string literal token.
- **Numtype**: Numeric types, like **i32**, **i64**, **f32** and **f64**, as well as **u32**, **i8**, **s16** *et cetera*.
- **Reftype**: One of **pointer** or **proxy**.
- **Valtype**: A **Numtype** or **Reftype**.
- **Identifier**: A user-assigned identifier (mapped to an index within an indexspace).
- **Identity**: A **Number** or **Identifier** (used to identify a component within its indexspace).

The first two pages of this document cover statements, while the last two pages cover instructions. Comments are used to explain things that would otherwise complicate the abstract grammar:

```
module : statement* ; always onside, each on its own line
```

```
statement : import "field" [from "module"] as component-specifier
          | export "field" as component-reference
          | define component-definition
```

```
component-specifier : function-specifier
                   | register-specifier
                   | memory-specifier
                   | table-specifier
```

```
component-reference : function-reference
                   | register-reference
                   | memory-reference
                   | table-reference
                   | memory-bank-reference
                   | table-bank-reference
                   | type-reference
```

```
component-definition : function-definition
                    | register-definition
                    | memory-definition
                    | table-definition
                    | memory-bank-definition
                    | table-bank-definition
                    | type-definition
```

```

function-reference      : [function] Identity
register-reference      : register Identity
memory-reference       : memory [Identity]                ; memories and tables default to zero
table-reference        : table [Identity]
memory-bank-reference  : memory bank Identity             ; banks never have defaults
table-bank-reference   : table bank Identity
type-reference         : type Identity

```

```

function-specifier : [start] function [of type]           ; types always default to `void → void`
                  | [start] Identifier [of type]
                  | [start] function Identifier [of type]
register-specifier : constant Valtype [Identifier]        ; registers must have a primitive valtype
                  | variable Valtype [Identifier]
memory-specifier   : [shared] memory [Identifier] limits
table-specifier    : table-qualifier table [Identifier] limits

```

```

function-definition : function-specifier function-block
register-definition  : register-specifier [initializer]    ; defaults to zero or null as appropriate
memory-definition   : memory-specifier [memory-primer]
table-definition    : table-specifier [table-primer]
memory-bank-definition : memory bank [Identifier] memory-bank-primer
table-bank-definition : table bank [Identifier] table-bank-primer
type-definition     : type [Identifier] as type-expression

```

```

table-qualifier : pointer | proxy | mixed

```

```

limits : with Number                ; max is implicitly equal to min
       | with Number to Number      ; min must never be more than max
       | with Number plus           ; not valid for shared memory limits

```

```

type : type-reference | type-expression | signature ; signatures are only valid in definitions

```

```

type-expression : types → types
signature       : params → types

```

```

types : Valtype,+ | void ; valtype is primitive
params : param,+ | void
param  : Valtype [Identifier] ; can be compounded, valtype is primitive

```

```

function-block      : register-preamble* [constant-expression]
constant-expression : instruction+ ; constant expressions can be inlined
memory-primer       : memory-segment+
table-primer        : table-segment+
memory-bank-primer  : datum+
table-bank-primer   : reference+

```

<pre> initializer : constant-expression with Number with Identity </pre>	<pre> ; numtype registers only, inline grammar ; pointer registers only, inline grammar </pre>
<pre> memory-segment : segment-preamble datum+ datum+ table-segment : segment-preamble reference+ reference+ </pre>	<pre> ; must be on its own line ; must be the first segment, can be inlined ; must be on its own line ; must be the first segment, can be inlined </pre>
<pre> register-preamble : @register Valtype [Identifier] segment-preamble : @segment constant-expression @segment Number </pre>	<pre> ; compoundable, valtype is primitive </pre>
<pre> datum : i8 Number i16 Number i32 Number i64 Number utf8 String reference : pointer Identity pointer null </pre>	<pre> ; encoding commands can be compounded ; reference commands can be compounded </pre>
<pre> instruction : unreachable nop return drop put Identity add Numtype sub Numtype mul Numtype div Numtype rem Numtype and Numtype or Numtype xor Numtype clz Numtype ctz Numtype abs Numtype neg Numtype min Numtype max Numtype floor Numtype nearest Numtype get [scope] Identity set [scope] Identity </pre>	<pre> ; type must be a float or gnostic integer ; type must be a gnostic integer ; type must be an integer ; type must be an integer ; type must be an integer ; type must be an integer ; type must be an integer ; type must be a float ; type must be a float ; type must be a float ; type must be a float ; type must be a float ; scope defaults to current scope ; scope defaults to current scope </pre>

```

instruction : rotate Numtype direction ; type must be an integer
| call Identity
| invoke type [in Identity]
| jump Identity
| fork Identity
| exit Identity+
| block [Identifier] [of type] constant-expression ; expression must be indented
| loop [Identifier] [of type] constant-expression ; expression must be indented
| branch [Identifier] [of type] constant-expression ; expression must be indented
| else constant-expression ; expression must be indented
| select [Valtype]
| grow writeable-type [Identity]
| size writeable-type [Identity]
| fill writeable-type [Identity]
| drop writeable-type Identity
| copy readable-type [to Identity]
| is positive-test
| not negative-test
| wrap i64 to i32
| promote f32 to f64
| demote f64 to f32
| convert Numtype to Numtype ; gnostic integer to float
| bitcast Numtype to Numtype ; float to integer | integer to float
| lop Numtype to Numtype ; float to gnostic integer
| lop Numtype sop Numtype ; float to gnostic integer
| extend Numtype ; 32-bit gnostic integer
| expand Numtype as Numtype ; integer as smaller signed integer
| load Numtype [as datatype] [in Identity] [at Number]
| store Numtype [as datatype] [in Identity] [at Number]
| atomic load Numtype [as datatype] [in Identity] [at Number]
| atomic store Numtype [as datatype] [in Identity] [at Number]
| atomic add Numtype [as datatype] [in Identity] [at Number]
| atomic sub Numtype [as datatype] [in Identity] [at Number]
| atomic and Numtype [as datatype] [in Identity] [at Number]
| atomic or Numtype [as datatype] [in Identity] [at Number]
| atomic xor Numtype [as datatype] [in Identity] [at Number]
| atomic trade Numtype [as datatype] [in Identity] [at Number]
| atomic broker Numtype [as datatype] [in Identity] [at Number]
| atomic wait Numtype [in Identity] [at Number]
| atomic notify [in Identity] [at Number]
| atomic fence
| shift-instruction
| push-instruction

```

```

shift-instruction : shift Numtype left ; integer
| shift Numtype right ; gnostic integer
push-instruction : push [Numtype] Number ; numtype defaults to i32
| push pointer Identity
| push Reftype null

```

direction : left | right
scope : local | global | table

writeable-type : memory | table
readable-type : writeable-type | memory bank | table bank

negative-test : more | less | equal
positive-test : negative-test | zero | null

datatype : i8 | i16 | i32
 | s8 | s16 | s32
 | u8 | u16 | u32