Anduril Reference Card

Layout based on *R Reference Card* by Tom Short, tshort@epri-peac.com.
Granted to the public domain. You can find more examples in the Anduril user guide.

Syntax

boolean
Assigned e.g. `var=true` or `var=false`.

int
Assigned e.g. `var=2`.

float
Assigned e.g. `var=2.00` or `var=4.53e-5`

string
Assigned e.g. `var="text"` or `var='text'`. When using double quotes, escaped characters can be used, such as \n for new line, \r for carriage return, \" for a quotation mark, \\ for a backslash, and \t for tabular. Note: when passing escaped characters to components - escape twice! To concatenate strings, use `var+"other string"`. Triple quoting can be used for multi line assignment: `var""" first line second line """`. Line feeds can be omitted with a single backslash at the end of a line. No escapes are used within triple single quotes.

record
Assign an empty record `var=record()` or equivalently `var={}`. Assign values to record: `var['key']="value"` or `var.key="value"`. Keys may be strings or integers.

include statement
Additional files of AndurilScript can be inlined with the `include 'filename.and'` statement. All file paths are kept relative to the script files using them.

if statement
`if (expression) { ...statements... } else {...}`
The expression may contain value comparisons, using operators: `&&, ||, !, ==, !=, <, <=, >, >=, +, -, *, /`. Parentheses "(" and ")" modify evaluation order.

Functions
```
function MyFunction(InType1 in1, ..., ParType1 param1)
-> (OutType1 out1, ..., OutTypeN outN)
{ ...statements...
return record(out1=x1, ..., outN=xN) }
```

Commenting
Java style comments include a row comment "`// comment `", a block comment "`/* comment */`" and documentation comment "`/** Doc... */`".

Environmental variables
Accesses an environmental variable with `$VARIABLENAME`. The script will fail if the variable is not set. Consider for instance `if std.exists('PATH',type='env') { std.echo($PATH) }`

Generic builtin functions

std.echo(any..., sep=" ")
Print each argument on screen. Argument may be strings, numbers, instances or any object.

std.time(value, in="yyyy-MM-dd HH.mm.ss", out="yyyy-MM-dd HH.mm.ss")
Converts current time or the given `value` to a string of the `out` format. Values may be given in `in` format, which follows the Java SimpleDateFormat syntax.

std.recordToString(record, valueSep="=", itemSep=",", keys=true, values=true)
Converts a record to a string. A key-value pair is separated with valueSep if both are included, whereas each two consecutive pairs are separated with `itemSep`. An empty string is produced if keys and values are both false.

std.fail(any..., assert=false, sep=" ")
Produces an error with the message that is constructed as in std.echo(). This function can be made conditional if assert parameter is used. Statements with `assert=true` are skipped silently and they can be used as invariants to confirm various conditions.

std.lookup(string)
Return the object (variable) whose name is given as a string or an error if the name is not found. This functions makes it possible to find component instances or other variables dynamically based on name.

std.convert(boolean/float/int/record, type=string)
Converts the given object to the requested type. Type is the name of the output type: boolean, float, int. For the records, each item is converted independently and a new record is produced for the output.

std.exists(string/record, type/key=string)
Returns a boolean indicating the existence of an environment variable (`env`), a file (`file`), an AndurilScript object (`object`), or an element (record and key pair) of the given name.

std.registerJava(string)
Register a custom native function that can be used like standard native functions.

std.metadata()
Returns a record with fields `instanceName` (component instance for an AndurilScript function call; not null if used within AndurilScript function), `Anduril` (engine version), `file` (source code AndurilScript pipeline filename), `path` (source code AndurilScript pipeline file absolute path), `location` (pretty print for location in source code where this std function is called), `line` (line location numerical value) and `column` (column location numerical value).

Iterators

for key,value: record { }
Iterate over the out-ports of a `record`. `key` is a string containing the record out-port name. `value` is the out-port itself, and can be used as an input for components.

for i: std.range(n,m,d) { }
Iterate `i` from int `n` to int `m` with an increment of `d`. The increment defaults to 1. `m` may be less than `n`, in which case the sequence is decreasing.

for s: std.split(text,d) { }
Split `text` using the delimiter `d`. The default delimiter accepts whitespaces.

for row: std.itercsv(input) {}
Iterate over a tab-delimited file. The `input` may be a string that points to a static file, or a CSV type component out-port/record, for dynamic for-loops. The `row` element contains strings matching the tab-delimited file headers. e.g. `row.Col1`

for file: std.iterdir(input, includeDir=true) {}
Iterate over a directory of files and if includeDir is true, also the directories. The `input` may be a string that points to a static directory, or a folder type component out-port/record, for dynamic for-loops. The `file` element contains two strings and a boolean: `file.name` for the base name of the file, `file.path` for absolute path to the file and `file.isDir` denoting whether the file is a directory.

for rec: std.iterArray(array) {}
Iterate over an Array type out-port. The `rec` element contains two strings: `rec.key` for the key name of the item, and `rec.file` for absolute path to the file. Array records may be returned with `array[rec.key]`.

std.enumerate(iterator)
Produce a higher order iterator, returning a numeric index for each iteration. e.g.
`for i,file: std.enumerate(std.iterdir(input)) { std.echo(i,file.name,file.path) }`

File functions

std.fRead(input)
Reads a file and returns it as a string. The `input` may be a string pointing to a static file, or an out-port from a component.

std.nRows(input)
Reads a tab-delimited file and returns its row count. The `input` may be a string pointing to a static file, or an out-port from a component. The header is not counted.

String functions

std.concat(any..., sep=" ")
Produces a concatenation of the string representations of the arguments. The given separator is used between the elements.

std.substring(string,n,m)
Returns a string from integer index `n` to index `m`. The indices start from 0.

std.length(string/record)
Returns the number of characters in a string or the number of elements in a record.

std.strReplace(string,match,replace,...)

Returns a string with all `match` replaced with string `replace`. The ellipsis, ..., can contain further match-replace pairs to avoid multiple calls to this function if they are not encapsulated into a record. The `match` may be a regular expression, for example: `std.strReplace("ilurand","(^..)(ur)(.*)","$3$2$1")`

`std.quote(string, type=string)`

Adds language specific escape sequences to the given string so that it can be embedded to the source codes based on the language. Possible types are: `Anduril`, `html`, `latex` and `url`.

Numeric functions

`std.mod(n,m)`

Return the remainder from the division of `n` by `m`.

`std.pow(a,b)`

Return the value of `n` to the power of `m`.

Array functions

`std.makeArray(record)`

Returns an array instance from a `record`.

Annotations

Collections of annotations

`@=record_name`

A record can map a set of annotation labels to valid values; such a record is used then to annotate a component instance or a port:

```
ci_annot = record(execute = "always", priority = 10)
port_annot = record(require = true)
ci1 = myCi(in1, @=port_annot in2, @=ci_annot)
ci2 = myCi(in1, @require=true in2, @execute="always")
```

Component annotations

Component annotations may be given as parameters to a component instance, e.g. `instance=Component(input,@annotation=value)`

`@bind=instance`

The annotated instances are made dependent on the execution of the assigned instance as if they would be dependent on its outputs.

`@enabled=boolean`

Default: `true`. If the value is `false`, the instance is never run.

`@execute=string`

Default: `changed`. Defines when a component is executed. By default all components are executed when their configuration is changed. Value `always` will execute the component every time the script is run. Value `once` will execute the component once. On subsequent runs, it is not re-executed even if configuration has been changed.

`@host=string`

Host ID of a remote host on which the component instance is remote-executed.

`@keep=boolean`

Default: `true`. Component's output files are stored after usage. If value is `false`, all output files created by the component are removed from the disk after they have been used by downstream components in the script.

`@name=string`

Renames the component instance in the workflow. The instance is available under both the new and original name in the script. This annotation is needed especially in for-loops.

`@par=record`

Component instance parameters can be given as a record of parameter name and value pairs. No default. Consider,

```
recParams = record(p1 = 10, p2 = "sample", p3 = true)
c1 = CompIns( @par = recParams, n = 10 )
c2 = CompIns( @par = recParams, n = 11 )
```

`@priority=int`

A higher value means the component will be executed before other components with lower values.

Port annotations

Multiple port annotations may be given as parameters to a port in a component instance, e.g. `instance=Component(@annotation=value input)`

`@require=boolean`

Default: `true`. Optional in-ports are treated as mandatory.