

---

# **Moore.io Client User Manual**

***Release 2.2.6***

**True North Silicon Inc.**

**Mar 08, 2026**



# CONTENTS

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Installation</b>	<b>3</b>
<b>3</b>	<b>Tools</b>	<b>5</b>
<b>4</b>	<b>Sample Session</b>	<b>7</b>
<b>5</b>	<b>Concepts</b>	<b>9</b>
<b>6</b>	<b>Commands</b>	<b>17</b>
<b>7</b>	<b>Configuration Space</b>	<b>25</b>
<b>8</b>	<b>API Documentation</b>	<b>35</b>



---

## OVERVIEW

The Moore.io Client is an open-source CLI tool designed to automate Electronic Design Automation (EDA) tasks encountered during the development of ASIC, FPGA and UVM IP. This tool also serves as a client for the Moore.io Web Site (<https://mooreio.com>), providing functionalities such as installing IP dependencies, generating UVM code and packaging/publishing IPs.

### 1.1 Why?

The EDA (Electronic Design Automation) field clearly lags behind in terms of Free & Open-Source (FOS) developer tools when compared to the software world. There is no FOS tool that can drive the various CAD software necessary and provide the kind of automation needed to produce commercial-grade FPGA and ASIC designs. Instead, homebrew (and seldom shared) Makefiles and Shell scripts rule the field of DevOps in semiconductors.

#### 1.1.1 The Problem

Writing a Makefile/Shell script that can perform all the tasks required to create a Chip Design is a LARGE job. Since these languages do not come with any meaningful libraries, the end result is a mess of patched, brittle code painful to maintain/debug, yet on which every engineering task directly depends. These issues are usually compounded by copying the project Makefile/Shell script from project to project; thus losing all Git history while commenting out old code and adding to the mess.

Surely, there must be a better way ...

#### 1.1.2 The Solution

The Moore.io Client is a FOS Command Line Interface (CLI) tool implemented in Python 3, using Object-Oriented Practices, strong typing, unit testing, and a modular architecture that will be very familiar to UVM engineers: the primary target audience of this tool. The Client, invoked via *mio*, has all the features you would expect from a “Project Makefile” at a high-end Semiconductor engineering firm AND all the best features from Software package managers:

- The concept of a Project, which is identified by a *mio.toml* file in its root directory
- A layered Configuration Space defined with TOML files (*mio.toml*) at the user (*~/.mio/mio.toml*) and project levels
- Packaging of HDL source file into IPs (Intellectual Property) identified by *ip.yml* descriptors in their root directory
- Performing tasks at the IP-level, including generating code, specifying and installing dependencies
- Ability to drive all Logic Simulators (VCS, XCelium, Questa, Vivado, Metrics DSim, Riviera-PRO) with a single set of commands and parameters
- A feature-driven Test Suite schema for specifying Regressions in UVM Test Benches, AND the ability to run these Regressions on Job Schedulers (LSF, GRID, etc.)

- Built-in compatibility with Continuous Integration (CI) tools like Jenkins

## 1.2 How much does it cost?

The EDA Automation and Package management is Free & Open-Source. Some commands, such as UVM Code Generation, “phone home” to the Moore.io Server and therefore require a User Account (which can be created at <https://mooreio.com/register>) and a license for Datum UVMx. However, the tool operates independently of the site in all other regards and can be used without authentication to build a Chip from start to finish.

## INSTALLATION

The Moore.io client is installed using `pip` (version  $\geq 3$ ):

```
pip install mooreio_client
```

To install/publish IP, generate UVM code as well as other features, a Moore.io User Account is required: <https://www.mooreio.com/register>





The Moore.io client currently supports the following tools, which must be installed separately:

- Metrics DSIm Desktop
- Metrics DSIm Cloud
- Xilinx Vivado
- Doxygen

To make these available, variables in the Configuration Space must be set:

- Altair DSIm: *altair\_dsim\_license\_path altair\_dsim\_installation\_path vscode\_installation\_path*
- Altair DSIm Cloud: *altair\_dsim\_cloud\_installation\_path*
- Xilinx Vivado: *xilinx\_vivado\_installation\_path*
- Doxygen: *doxygen\_installation\_path*

## 3.1 Example User Configuration

Save the following to `~/.mio/mio.toml`:

```
[logic_simulation]
default_simulator="dsim"
altair_dsim_license_path=~/.metrics-ca/dsim-license.json"
altair_dsim_installation_path="/tools/altair/dsim"
altair_dsim_cloud_installation_path="/usr/local/bin"
xilinx_vivado_installation_path="/tools/vivado/2024.2/Vivado/2024.2"
compilation_timeout=60.0
compilation_and_elaboration_timeout=60.0
elaboration_timeout=60.0
simulation_timeout=60.0

[applications]
editor="emacs"
web_browser="firefox"

[docs]
doxygen_installation_path="/usr/bin"
```



## SAMPLE SESSION

This section outlines a command listing that should enable you to get an overview of the `mio` workflow and its essential features. This example assumes you are familiar with CLI basics.

### 4.1 0. Clone the sample session repository

1. The repository is available from GitHub: [https://github.com/Datum-Technology-Corporation/mooreio\\_client\\_sample\\_session](https://github.com/Datum-Technology-Corporation/mooreio_client_sample_session)

```
git clone https://github.com/Datum-Technology-Corporation/
mooreio_client_sample_session.git mio_sample_session
```

### 4.2 1. Check installation and view Documentation

1. Ensure `mio` is installed and operational. Display basic help text:

```
mio --help
```

2. Print the CLI manual for a command:

```
mio help init
```

### 4.3 2. Initialize a new Project

1. Move into Project root directory:

```
cd mio_sample_session
```

2. Initialize a new Project. Creates `mio.toml` Project descriptor file and directory structure (if not present):

```
mio init
```

### 4.4 3. Initialize Design IP

1. Move into root directory of a design:

```
cd rtl/example_design
```

2. Initialize design IP: creates `ip.yml` IP descriptor file:

```
mio init
```

## 4.5 4. Initialize Test Bench IP

1. Move into root directory of a test bench:

```
cd dv/uvmt_example
```

2. Initialize test bench IP. Creates `ip.yml` IP descriptor and `ts.yml` Test Suite for defining regressions:

```
mio init
```

## 4.6 5. Install IP dependencies

1. Install the entire dependency tree from <https://www.mooreio.com> for a specific IP:

```
mio install uvmt_example
```

## 4.7 6. Run a simulation

1. Run a test for a UVM test bench IP we initialized.

```
mio sim uvmt_example -t smoke -s 1 -w -c -a dsim
```

- IP: `uvmt_example`
- Test Name: `smoke`
- Seed: `1`
- Waveform capture: `Enabled`
- Coverage sampling: `Enabled`
- Simulator: `Altair DSim`

2. View simulation results.

All commands for viewing the results of a single test run are printed automatically to the terminal after simulation has ended.

## 4.8 7. Run a regression

1. Clean up a work area. (Optional)

```
mio clean uvmt_example
```

Deletes all compiled HDL code for block test bench, including external dependencies. Whenever you encounter unexpected elaboration errors, run an `mio clean` command on your IP and try again. Does not delete results.

2. Run sanity regression.

```
mio regr uvmt_example sanity
```

- Runs *uvmt\_example*'s Test Suite's "sanity" regression.
- See command output for generated report locations (similar to simulation results).

## CONCEPTS

## 5.1 Project

A Project is a collection of IPs and the directories where simulation, linting, synthesis and other EDA task results are stored. Only one Project is active at a time. The working directory from which `mio` is invoked must either be the root of the Project or within a sub-directory.

A Project Descriptor file (`mio.toml`) in the root directory defines metadata specific for the Project. This file must contain the Project name, `full_name` and `sync`. The latter denotes synchronization with the Moore.io server. The contents of this file are part of the Configuration Space and can set any parameter therein.

### 5.1.1 Sample Descriptor

```
[project]
sync      = false
name      = "my_chip"
full_name= "My Chip"

[ip]
local_paths=["rtl", "dv"]

[logic_simulation]
root_path="sim"
uvm_version="1.2"

[docs]
root_path="docs"
```

## 5.2 Configuration Space

The Configuration space is loaded from multiple `mio.toml` files which are merged into a final configuration space. This information is then used in all further MIO operations. The full Configuration Space is detailed in its own section.

### 5.2.1 Directory Structure

Moore.io stores its internal files under `.mio` in the Project root directory. These files are to be considered read-only and should not be modified. Any changes can lead to unpredictable behavior. It is highly recommended to add this directory to your Project `.gitignore`.

IPs are by default stored under `rtl` and `dv`. These names are arbitrary and there is no obligation to store a particular type of IP in one or the other.

Each EDA task has its own directory for results and reports, here are the defaults:

- Logic Simulation - `sim`
- Documentation - `docs`

Coming soon:

- Linting - `lint`
- Synthesis - `syn`
- Design for test - `dft`
- Formal verification - `fml`
- Place and route - `pnr`
- Physical layout - `lay`

## 5.3 IP

An IP is a described collection of source code, directories, and associated documentation files. This descriptor is captured using YAML and has a standard structure, regardless of its content.

### 5.3.1 Structure

The standard structure is defined below:

- `ip` - Datasheet
  - `sync` - *Boolean* - Synchronized with <https://www.mooreio.com>
  - `pkg_type` - *Enum* - IP Type [`dv_lib`, `dv_agent`, `dv_env`, `dv_tb`, `lib`, `block`, `ss`, `fpga`, `chip`, `system`, `custom`]
  - `vendor` - *String* - (Short) Name of the Organization/User owning this IP.
  - `name` - *String* - (Short) Name
  - `full_name` - *String* - Descriptive name
  - `version` - *String* - Current version
  - `sync_id` - *Int* - (Optional) ID with <https://www.mooreio.com>
  - `sync_revision` - *String* - (Optional) Current synced version
  - `encrypted` - *List* - (Optional) List of simulators for which this IP provides encrypted HDL source
  - `mlicensed` - *Boolean* - (Optional) Indicates a license is required this IP
- `structure` - Paths for directory structure.
  - `hdl_src_path` - *String* - Location of HDL source code (SystemVerilog/VHDL).
  - `docs_path` - *String* - Location of documents for Doxygen to use when generating its reference documentation contents. Articles can be added by adding markdown (.md) files in this directory.
  - `scripts_path` - *String* - Location of scripts and misc. files.
  - `examples_path` - *String* - Location of sample code for users.
- `hdl_src` - Describes Hardware Description Language source code structure and data for compilation and simulation.

- `directories` - *List* - Paths where source code is located, relative to `src-path`. “.” equates the HDL source directory root.
- `top_sv_files` - *List* - (Optional) Path to top SystemVerilog source file(s) for compilation.
- `top_vhdl_files` - *List* - (Optional) Path to top VHDL source file(s) for compilation.
- `top` - *List* - Name of top-level HDL module(s) for elaboration.
- `so_libs` - *List* - Name of DPI Shared Objects `.so` to be loaded. These must be located in the `scripts_path` directory of the IP. Filename convention is `<name>.<simulator>.so`.
- `tests_path` - *String* - Path to tests source code files. *Test Bench IPs only*.
- `tests_name_template` - *String* - Jinja2 template with single argument: `name` describing the test naming convention for this IP. *Test Bench IPs only*.
- `dependencies` - *List* - (Optional) IP dependencies. Can include both local (project) and external (from Moore.io IP catalog).
  - `"<Vendor>/<IP Name>" : "<VersionSpec>"` - Where `<IP Name>` is the (short) name of an IP and `<VersionSpec>` is the required version of that IP via a semantic version spec.
- `dut` - (Optional) For Test Bench (`dv_tb`) IPs, the section describes the Device Under Test.
  - `type` - *Enum* - The DUT type (additional types coming soon): `[ip]`
  - `name` - *String* - IP (Short) Name.
  - `version` - *VersionSpec* - Semantic version spec for the DUT
  - `target` - *String* - Default target for the DUT
- `targets` - (Optional) List of IP Targets.
  - `<Name>` : - Target name. `default` is used when a target is not provided by the user
    - \* `dut` - *String* - DUT target name
    - \* `cmp` - *List* - Compilation arguments: *String* and *Boolean* are valid.
    - \* `elab` - *List* - Elaboration arguments (coming soon).
    - \* `sim` - *List* - Simulation arguments: *String* and *Boolean* are valid.

### 5.3.2 RTL IP Sample Descriptor

The following is an IP descriptor for an Ethernet 100G MAC:

```
ip:
  sync      : false
  pkg_type  : ss
  vendor    : acme
  name      : eth_mac_100g
  full-name : 100G Ethernet MAC
  version   : 2.1.4

dependencies:
  "datron/fec_lib": ">=1.2"
  "gigamicro/pcs_encoder": "5.1"

structure:
  src-path: "src"
```

(continues on next page)

(continued from previous page)

```

hdl-src:
  directories: [".", "mac", "pcs", "fec"]
  top_sv_files: ["eth_100g_top.sv"]

targets:
  default:
    cmp:
      PMA_WIDTH: 32
      INCLUDE_FEC: true
    fast:
      cmp:
        INCLUDE_FEC: false
    wide:
      cmp:
        PMA_WIDTH: 64

```

### 5.3.3 Test Bench IP Sample Descriptor

The following is an IP descriptor for a Test Bench called '100G Ethernet MAC Sub-System':

```

ip:
  sync      : false
  pkg_type  : dv_tb
  vendor    : acme
  name      : uvmt_eth_mac_100g
  full_name : 100G Ethernet MAC Sub-System Test Bench
  version   : 1.1.2

dut:
  type: ip
  name: "acme/eth_mac_100g"
  version: *

structure:
  scripts_path : "bin"
  docs_path    : "docs"
  examples_path: "examples"
  src_path     : "src"

hdl_src:
  directories      : ["."]
  top_sv_files     : ["uvmt_eth_mac_100g_pkg.sv"]
  top              : ["uvmt_eth_mac_100g_tb"]
  tests_path       : "tests"
  tests_name_template: "uvmt_eth_mac_100g_{{ name }}_test_c"

targets:
  default:
    dut: fast
    sim:
      NUM_PKTS: 10

```

(continues on next page)



(continued from previous page)

```
release:
  dut: default
  sim:
    NUM_PKTS: 1000
```

## 5.4 IP Marketplace

The Moore.io IP Marketplace hosts the IP catalog and its source code. Developers interact with the marketplace primarily via `mio login`, `mio install` and `mio publish`

### 5.4.1 License Types

- Free & Open Source (FOS) - Free to list. Source code and documentation stored on the Marketplace.
- Commercial - IPs that use the Moore.io IP Licensing System to charge end-user for IP.
- Private - Use Moore.io as your Private IP server; ideal for clean rooms and sites with restricted internet access. Coming soon.

## 5.5 Test Suite

Moore.io's regression system flips the script on the usual regression bash scripts of old. Instead, Test Suite descriptors (`ts.yml` for default (single) target IPs, `<Name>.ts.yml` for multiple target IPs) describe several regressions at once, with an inside-out approach of Test Sets and Test Groups.

### 5.5.1 Structure

All test suites have 2 sections. The metadata `ts` and the regression definitions `tests`. `mio` does not currently interface with scheduling engines such as GRID or LSF, but plans to in the near future.

The `max_duration` feature allows `mio` to prematurely end regressions via a hard time limit. Simulations processes are simply killed off.

### 5.5.2 Metadata

This section of the test suite contains the information necessary to run the regressions.

- `ts` - Test Suite Metadata
  - `name` - *String* - Descriptive name. Ex: "Data Sub-System Test Suite for APB interconnect"
  - `ip` - *String* - Owner IP. Ex: "uvmt\_data\_ss"
  - `target` - *List[String]* - Target Name. Ex: "apb"
  - `waves` - *String[]* - List of regressions for which wave capture is enabled. Ex: [sanity, bugs]
  - `cov` - *String[]* - List of regressions for which coverage sampling is enabled. Ex: [nightly, weekly]
  - `verbosity` - *String[String]* - Dictionary mapping each regression with a UVM logging verbosity level. Ex: {sanity:high, nightly:medium}
  - `max_duration` - *Integer[Float]* - Dictionary mapping each regression with a timeout (specified in hours). Ex: {sanity:1, nightly:5}
  - `max_jobs` - *Integer[Integer]* - Dictionary mapping each regression with a limit on concurrent simulations. Ex: {sanity:5, nightly:10}

### 5.5.3 Regressions

This section of the test suite defines the contents of the regressions.

- tests
  - Test Set - Ex: `functional` - Top-level elements; encapsulate test groups.
    - \* Test - Ex: `reg_bit_bash` - The name of the test used must match what would be entered on the command line.
      - Regression Entry - *String* - Regression name. Ex: `sanity`

### 5.5.4 Regression Entries

A regression entry can be either:

- *Int* - Amount of random seeds
- *List[Int]* - Specific seeds
- Default Group
- Named Groups

A Group is a pairing of seeds and args for specifying simulation arguments:

```
smoke_test:
  group_a:
    seeds: 10
    args:
      ABC: 100
      DEF: false
```

### 5.5.5 Sample

```
ts:
  name: Default
  ip: uvmt_eth_mac_100g
  target: ["*"]
  waves: [sanity, bugs]
  cov : [nightly, weekly]
  max_errors : { 'sanity': 1, 'nightly': 30, 'weekly': 30, 'bugs': 1 }
  verbosity : { 'sanity': 'high', 'nightly': 'medium', 'weekly': 'low', 'bugs': 'debug' }
  max_duration: { 'sanity': .25, 'nightly': 5, 'weekly': 12, 'bugs': 1 }
  max_jobs : { 'sanity': 5, 'nightly': 10, 'weekly': 20, 'bugs': 1 }

tests:
  functional:
    rand_traffic:
      sanity : [1]
      nightly: 10
      weekly : 50
    weekly :
```

(continues on next page)

(continued from previous page)

```
seeds: 50
args :
  NUM_PKTS: 1000
bugs: [5456984247]
reg_hw_reset:
  sanity: [1]
  nightly: 1
  weekly: 1
reg_bit_bash:
  sanity: [1]
  nightly: 1
  weekly: 1

error:
  rand_err_traffic:
    sanity : [1]
    nightly: 10
    weekly :
      small_packets:
        seeds: 5
        args :
          MIN_PKT_SZ: 64
          MAX_PKT_SZ: 127
      large_packets:
        seeds: 5
        args :
          MIN_PKT_SZ: 256
          MAX_PKT_SZ: 512
  bugs: [
    8438499331868,
    6424554831489
  ]
```



## COMMANDS

All commands can be prepended by `--dbg` to enable mio's debug printout. Locations for custom commands are specified via the `MIO_CUSTOM_COMMANDS` environment variable.

Custom commands must be in Python3 and extend from a *Command* class. Ex:

```
from mio_client.commands.sim import SimulateCommand

class MySimulateCommand(SimulateCommand):
    def phase_main(self, phase: Phase):
        self.rmh.info("Hello, World!")
        super().phase_main(phase)

    def compile(self, phase: Phase):
        super().compile(phase)
        # ...

    def elaborate(self, phase: Phase):
        super().elaborate(phase)
        # ...

    def simulate(self, phase: Phase):
        super().simulate(phase)
        # ...

    def phase_report(self, phase: Phase):
        super().phase_report(phase)
        self.rmh.info(f"Custom report:\n# compilation warnings: {self.compilation_report.
↪num_warnings}")
```

## 6.1 Credentials Management

### 6.1.1 login

#### Description

Authenticates session with Moore.io Server.

## Usage

`mio login [OPTIONS]`

## Options

<code>-u USERNAME</code>	<code>--username USERNAME</code> Specifies Moore.io username
<code>-n</code>	<code>--no-input</code> Specify credentials without a keyboard. Must be combined with <code>-u</code> and by setting the environment variable <code>MIO_AUTHENTICATION_PASSWORD</code>

## Examples

<code>mio login</code>	Log in with prompts for username and password
<code>mio login -u user123</code>	Specify username inline and only get prompted for the password
<code>mio login -u user123 --no-input</code>	Authenticate without a keyboard (especially handy for CI)

### 6.1.2 logout

#### Description

De-authenticates session with Moore.io Server.

#### Usage

`mio logout`

## 6.2 Documentation

### 6.2.1 dox

#### Description

Generates reference documentation from IP HDL source code using Doxygen.

#### Usage

`mio dox [IP]`

#### Examples

<code>mio dox my_ip</code>	Generates HTML documentation for IP <code>my_ip</code>
<code>mio dox</code>	Generates HTML all local Project IPs

### 6.2.2 help

#### Description

Prints out documentation on a specific command.

## Usage

`mio help CMD`

## Examples

<code>mio help sim</code>	Prints out a summary on the Logic Simulation command and its options
---------------------------	--

## 6.3 EDA

### 6.3.1 clean

#### Description

Deletes output artifacts from EDA tools and/or Moore.io Project directory contents `.mio`. Only logic simulation artifacts are currently supported.

#### Usage

`mio clean [IP] [OPTIONS]`

#### Options

<code>-d</code>	<code>--deep</code>	Removes Project Moore.io directory <code>.mio</code>
-----------------	---------------------	--

## Examples

<code>mio clean my_ip</code>	Delete compilation, elaboration and simulation artifacts for IP 'my_ip'
<code>mio clean --deep</code>	Removes contents of Project Moore.io directory <code>mio</code>

### 6.3.2 sim

#### Description

Performs necessary steps to run simulation of an IP. Only supports Digital Logic Simulation for the time being.

An optional target may be specified for the IP. Ex: `my_ip#target`.

While the controls for individual steps (DUT setup, compilation, elaboration and simulation) are exposed, it is recommended to let *mio sim* manage this process as much as possible. In the event of corrupt simulator artifacts, see *mio clean*. Combining any of the step-control arguments (`-D`, `-X`, `-C`, `-E`, `-S`) with missing steps is illegal (ex: `-DS`).

Two types of arguments (`--args`) can be passed: compilation (`+define+NAME[=VALUE]`) and simulation (`+NAME[=VALUE]`).

For running multiple tests in parallel, see `mio regr`.

#### Usage

`mio sim IP[#TARGET] [OPTIONS] [--args ARG ...]`

## Options

-t TEST	--test TEST	Specify the UVM test to be run.
-s SEED	--seed SEED	Positive Integer. Specify randomization seed. If none is provided, a random one will be picked.
-v VERBOSITY	--verbosity VERBOSITY	Specifies UVM logging verbosity: none, low, medium, high, full, debug. [default: medium]
-+ ARGS	--args ARGS	Specifies compilation-time (+define+ARG[=VAL]) or simulation-time (+ARG[=VAL]) arguments
-e ERRORS	--errors ERRORS	Specifies the number of errors at which compilation/elaboration/simulation is terminated. [default: 10]
-a APP	--app APP	Specifies simulator application to use: dsim, vivado.
-w	--waves	Enable wave capture to disk.
-c	--cov	Enable code & functional coverage capture.
-g	--gui	Invokes simulator in graphical or 'GUI' mode.
-d DEST	--dry-run DEST	Captures simulation command into tarball at DEST instead of invoking simulator.

## Steps

-D	Prepare Device-Under-Test (DUT) for logic simulation. Ex: invoke FuseSoC to prepare core(s) for compilation.
-X	Invoke Datum UVMx for code generation.
-C	Compile
-E	Elaborate
-S	Simulate

## Examples

mio sim my_ip -t smoke -s 1 -w -c	Compile, elaborate and simulate test my_ip_smoke_test_c for IP my_ip with seed 1 and waves & coverage capture enabled.
mio sim my_ip -t smoke -s 1 --args +NPKTS=10	Compile, elaborate and simulate test my_ip_smoke_test_c for IP my_ip with seed 1 and a simulation argument.
mio sim my_ip -S -t smoke -s 42 -v high -g	Only simulates test my_ip_smoke_test_c for IP my_ip with seed 42 and UVM_HIGH verbosity using the simulator in GUI mode.
mio sim my_ip -C	Only compile my_ip.
mio sim my_ip -E	Only elaborate my_ip.
mio sim my_ip -CE	Compile and elaborate my_ip.

### 6.3.3 regr

#### Description

Runs a regression (set of tests) against a specific IP. Regressions are described in Test Suite files ([<target>].ts.yml).

An optional target may be specified for the IP. Ex: my\_ip#target.



## Usage

```
mio regr IP[#TARGET] [TEST SUITE.]REGRESSION [OPTIONS]
```

## Options

-a	--app	Specifies which simulator to use: dsim, dsimc, vivado.
-d	--dry-run	Compiles, elaborates, but only prints the tests mio would normally run (does not actually run them).

## Examples

<code>mio regr my_ip sanity -a dsim</code>	Run sanity regression for IP <code>uvm_my_ip</code> , from test suite <code>ts.yml</code> using Altair DSIm
<code>mio regr my_ip apb_xc.sanity</code>	Run sanity regression for IP <code>uvm_my_ip</code> , from test suite <code>apb_xc.ts.yml</code> using the default simulator
<code>mio regr my_ip axi_xc.sanity -d -a dsimc</code>	Dry-run sanity regression for IP <code>uvm_my_ip</code> , from test suite <code>axi_xc.ts.yml</code> using Altair DSIm Cloud

# 6.4 Generators

## 6.4.1 init

### Description

Creates a new Project skeleton if not already within a Project. If so, a new IP skeleton is created. This is the recommended method for importing code to the Moore.io ecosystem.

### Usage

```
mio init [OPTIONS]
```

## Options

-i	--input-file	Specifies YAML input file path (instead of prompting user)
----	--------------	--

## Examples

<code>mio init</code>	Create a new empty Project/IP in this location.
<code>mio init -i ~/answers.yml</code>	Create a new empty Project/IP in this location with pre-filled data.
<code>mio -C ~/my_proj init</code>	Create a new empty Project at a specific location.

## 6.4.2 uvmx

### Description

Generates IP HDL code using Datum UVMx (requires license). If not within an initialized Project, the ID must be specified via `-p/--project-id`.

## Usage

`mio x [OPTIONS]`

## Options

<code>-p ID</code>	<code>--project-id=ID</code>	Specifies Project ID when initializing a new project
<code>-f</code>	<code>--force</code>	Overwrites user changes

## Examples

<code>mio x</code>	Sync (generate) project with UVMx definition on server
<code>mio x -p 123</code>	Initialize and generate Project from empty directory

# 6.5 IP Management

## 6.5.1 install

### Description

Downloads IP(s) from Moore.io Server. Can be used in 3 ways:

1. Without specifying an IP: install all missing dependencies for all IPs in the current Project
2. Specifying the name a local IP: install all missing dependencies for a specific IP in the current project
3. Specifying the name of an IP on the Moore.io Server: install remote IP and all its dependencies into the current Project

### Usage

`mio install [IP] [OPTIONS]`

### Options

<code>-v</code>	<code>--version</code>	Specifies IP version (only for remote IPs). Must specify IP when using this option.
<code>SPEC</code>	<code>SPEC</code>	

## Examples

<code>mio install</code>	Install all dependencies for all IPs in the current Project
<code>mio install my_ip</code>	Install all dependencies for a specific IP in the current Project
<code>mio install acme/abc</code>	Install latest version of IP from Moore.io Server and its dependencies into current Project
<code>mio install acme/abc -v "1.2.3"</code>	Install specific version of IP from Moore.io Server and its dependencies into current Project

## 6.5.2 uninstall

### Description

Removes IP(s) installed in current Project. Can be used in 3 ways:

1. Without specifying an IP: delete all installed dependencies for all IPs in the current Project
2. Specifying the name a local IP: delete all installed dependencies for a specific local IP in the current project
3. Specifying the name of an installed IP: delete installed IP and all its installed dependencies from the current Project

### Usage

```
mio uninstall [IP]
```

### Examples

<code>mio uninstall</code>	Delete all installed IPs in current project
<code>mio uninstall my_ip</code>	Delete all installed dependencies for a specific local IP in the current project
<code>mio install acme/abc</code>	Delete specific installed IP and all its installed dependencies from current project

## 6.5.3 package

### Description

Command for encrypting/compressing entire IP on local disk. To enable IP encryption, add an ‘encrypted’ entry to the `hdl_src` section of your descriptor (`ip.yml`). Moore.io will only attempt to encrypt using the simulators listed under ‘encrypted’ of the ‘ip’ section.

### Usage

```
mio package IP DEST
```

### Examples

```
mio package uvma_my_ip ~ Create compressed archive of IP uvma_my_ip under user's home directory.
```

## 6.5.4 publish

### Description

Packages and publishes an IP to the Moore.io IP Marketplace (<https://mooreio.com>). Currently only available to administrator accounts.

### Usage

```
mio publish IP [OPTIONS]
```

### Options

```
-c ORG --customer ORG Specifies Customer Organization name. Commercial IPs only.
```

## Examples

<code>mio publish uvma_my_ip</code>	Publish Public IP 'uvma_my_ip'.
<code>mio publish uvma_my_ip -c acme</code>	Publish Commercial IP 'uvma_my_ip' for customer 'acme'.

## CONFIGURATION SPACE

`mio` behavior is controlled by the configuration space, which is obtained from multiple sources, in decreasing precedence:

- Command line parameters - `-c <name>=<value>` or `--config=<name>=<value>` - *Coming soon*
- Project configuration file - `<PROJECT ROOT PATH>/mio.toml`
- User configuration file - `~/.mio/mio.toml`
- Built-in defaults - `<MOOREIO CLIENT INSTALLATION PATH>/data/mio.toml`

### 7.1 applications

#### 7.1.1 editor

- Required: Yes
- Type: Path
- Default: `vim`

Default text editor.

#### 7.1.2 web\_browser

- Required: Yes
- Type: Path
- Default: `firefox`

Default web browser.

### 7.2 authentication

#### 7.2.1 offline

- Required: Yes
- Type: Boolean
- Default: `false`

Prohibits `mio` from attempting to authenticate with the Moore.io Server.

## 7.2.2 server\_url

- Required: Yes
- Type: String
- Default: `https://mooreio.com`

Moore.io Server URL.

## 7.2.3 server\_api\_url

- Required: Yes
- Type: String
- Default: `https://mooreio.com/api`

Moore.io API Server URL.

## 7.3 docs

### 7.3.1 doxygen\_installation\_path

- Required: No
- Type: Path

Path to doxygen installation directory.

### 7.3.2 root\_path

- Required: Yes
- Type: Path
- Default: `docs`

Directory for Project documents. Doxygen outputs to `./doxygen_output` under this directory.

## 7.4 encryption

### 7.4.1 altair\_dsim\_sv\_key\_path

- Required: No
- Type: Path
- Example: `/tools/dsim_sv.key`

Absolute path to location of Altair DSIM SystemVerilog encryption key file. This must be set in order to encrypt IP using DSIM. See <https://help.Altair.ca/support/solutions/articles/154000141181-user-guide-dsim-ieee-1735-encryption-verilog-and-vhdl> for more information.

### 7.4.2 altair\_dsim\_vhdl\_key\_path

- Required: No
- Type: Path
- Example: `/tools/dsim_vhdl.key`

Absolute path to location of Altair DSIm VHDL encryption key file. This must be set in order to encrypt IP using DSIm. See <https://help.Altair.ca/support/solutions/articles/154000141181-user-guide-dsim-ieee-1735-encryption-verilog-and-vhdl-> for more information.

### 7.4.3 xilinx\_vivado\_key\_path

- Required: Yes
- Type: Path
- Example: /tools/viv.key

Absolute path to location of Xilinx Vivado encryption key file. This must be set in order to encrypt IP using vivado. See <https://www.xilinx.com/products/intellectual-property/ip-encryption.html> for more information.

## 7.5 ip

### 7.5.1 global\_paths

- Required: Yes
- Type: List[Path]
- Default: []

*mio* searches these absolute paths for IP descriptors.

### 7.5.2 local\_paths

- Required: Yes
- Type: List[Path]
- Default: ["dv", "rtl"]

*mio* searches these relative (to the project root) paths for IP descriptors. The names used are irrelevant to the IP types contained therein. Ex: DV IPs could be stored under `rtl` and vice-versa with no impact on functionality.

## 7.6 lint

### 7.6.1 root\_path

- Required: Yes
- Type: Path
- Default: lint

Project-relative path to directory where HDL linting results and reports are stored.

## 7.7 project

### 7.7.1 sync

- Required: Yes
- Type: Boolean
- Default: false

Denotes synchronization with the Moore.io Server.

### 7.7.2 sync\_id

- Required: No
- Type: Int

Synchronization ID with the Moore.io Server. Only present when sync is true.

### 7.7.3 local\_mode

- Required: Yes
- Type: Boolean
- Default: false

Prohibits mio from attempting to make HTTP requests.

### 7.7.4 name

- Required: Yes
- Type: String
- Example: chip\_123

Short name for the current project. Cannot contain spaces.

### 7.7.5 full\_name

- Required: Yes
- Type: String
- Example: Chip 123

Descriptive name for the current project.

### 7.7.6 description

- Required: No
- Type: String
- Example: Chip for 123 clients

Descriptive text for the current project.

## 7.8 logic\_simulation

### 7.8.1 compilation\_timeout

- Required: Yes
- Type: Float
- Default: 1.0

Timeout for compilation jobs. Measured in hour(s).



### 7.8.2 compilation\_and\_elaboration\_timeout

- Required: Yes
- Type: Float
- Default: 1.0

Timeout for compilation+elaboration jobs. Measured in hour(s).

### 7.8.3 default\_simulator

- Required: No
- Type: String

Simulator used when invoking the `sim` command without specifying `-a APP --app APP`.

### 7.8.4 elaboration\_timeout

- Required: Yes
- Type: Float
- Default: 1.0

Timeout for elaboration jobs. Measured in hour(s).

### 7.8.5 logs\_directory

- Required: Yes
- Type: String
- Default: results

Name of directory where compilation and elaboration results are output. This directory is always created directly under `root_path`.

### 7.8.6 altair\_dsim\_default\_compilation\_and\_elaboration\_arguments

- Required: Yes
- Type: List[String]
- Default: `["+acc+b", "-suppress MultiBlockWrite:ReadingOutputModport", "-warn UndefinedMacro:DupModuleDefn"]`

Compilation arguments always passed to Altair DSim during compilation+elaboration.

### 7.8.7 altair\_dsim\_default\_compilation\_sv\_arguments

- Required: Yes
- Type: List[String]
- Default: `["-suppress MultiBlockWrite:ReadingOutputModport:UndefinedMacro"]`

Compilation arguments always passed to Altair DSim during SystemVerilog compilation.

### 7.8.8 altair\_dsim\_default\_compilation\_vhdl\_arguments

- Required: Yes
- Type: List[String]
- Default: []

Compilation arguments always passed to Altair DSim during VHDL compilation.

### 7.8.9 altair\_dsim\_default\_elaboration\_arguments

- Required: Yes
- Type: List[String]
- Default: ["+acc+b", "-suppress DupModuleDefn"]

Compilation arguments always passed to Altair DSim during elaboration.

### 7.8.10 altair\_dsim\_default\_simulation\_arguments

- Required: Yes
- Type: List[String]
- Default: []

Compilation arguments always passed to Altair DSim during simulation.

### 7.8.11 altair\_dsim\_license\_path

- Required: No
- Type: Path

Path to Altair DSim Desktop license key.

### 7.8.12 altair\_dsim\_cloud\_installation\_path

- Required: No
- Type: Path

Path to Altair DSim Cloud simulator installation directory.

### 7.8.13 altair\_dsim\_installation\_path

- Required: No
- Type: Path

Path to Altair DSim Desktop installation directory.

### 7.8.14 root\_path

- Required: Yes
- Type: Path
- Default: sim

Project-relative path to directory where HDL simulations results and reports are stored.

### 7.8.15 regression\_directory\_name

- Required: Yes
- Type: String
- Default: regr

Name of directory where regressions results are stored. This directory is always created directly under `root_path`.

### 7.8.16 results\_directory\_name

- Required: Yes
- Type: String
- Default: results

Name of directory where immediate results are stored. This directory is always created directly under `root_path`.

### 7.8.17 simulation\_timeout

- Required: Yes
- Type: Float
- Default: 1.0

Timeout for simulation jobs. Measured in hour(s).

### 7.8.18 test\_result\_path\_template

- Required: Yes
- Type: String
- Default: `{{ ip }}{{ target }}_{{ test }}_{{ seed }}{% if args %}_ {% for arg in args %}{{ arg }}_ {% endfor %}{% endif %}`

Jinja2 template used to generate the directory names for IP simulation test results.

### 7.8.19 timescale

- Required: Yes
- Type: String
- Default: 1ns/1ps

Simulation timescale specified to the simulator via command line.

### 7.8.20 uvm\_version

- Required: Yes
- Type: String
- Default: 1.2

Specifies the version of UVM to be used during simulation.

### 7.8.21 `vscode_installation_path`

- Required: No
- Type: Path

Path to Microsoft VSCode installation directory. Used by `dsim` to view `.mxd` waveform files.

### 7.8.22 `xilinx_vivado_default_compilation_sv_arguments`

- Required: Yes
- Type: `List[String]`
- Default: `["--incr"]`

Compilation arguments always passed to Xilinx Vivado during SystemVerilog compilation.

### 7.8.23 `xilinx_vivado_default_compilation_vhdl_arguments`

- Required: Yes
- Type: `List[String]`
- Default: `[]`

Compilation arguments always passed to Xilinx Vivado during VHDL compilation.

### 7.8.24 `xilinx_vivado_default_elaboration_arguments`

- Required: Yes
- Type: `List[String]`
- Default: `["--incr", "-relax", "--00", "-dup_entity_as_module"]`

Compilation arguments always passed to Xilinx Vivado during elaboration.

### 7.8.25 `xilinx_vivado_default_simulation_arguments`

- Required: Yes
- Type: `List[String]`
- Default: `["--stats"]`

Compilation arguments always passed to Xilinx Vivado during simulation.

### 7.8.26 `xilinx_vivado_installation_path`

- Required: No
- Type: Path

Path to Xilinx Vivado installation directory.

## 7.9 `logic_synthesis`

### 7.9.1 `root_path`

- Required: Yes
- Type: Path

- Default: syn

Project-relative path to directory where logic synthesis results and reports are stored.

## 7.10 package\_management

### 7.10.1 fsoc\_cores\_global\_paths

- Required: Yes
- Type: List[Path]
- Default: []

FuseSoc searches these absolute paths for core files.

### 7.10.2 fsoc\_cores\_local\_paths

- Required: Yes
- Type: List[Path]
- Default: ["dv", "rtl"]

FuseSoc searches these relative (to the project root) paths for core files.



## API DOCUMENTATION

### 8.1 mio\_client package

#### 8.1.1 Subpackages

`mio_client.commands` package

##### Submodules

`mio_client.commands.gen` module

`mio_client.commands.ip` module

`mio_client.commands.misc` module

`mio_client.commands.sim` module

`mio_client.commands.user` module

##### Module contents

`mio_client.core` package

##### Submodules

`mio_client.core.command` module

`mio_client.core.configuration` module

`mio_client.core.ip` module

`mio_client.core.model` module

`mio_client.core.phase` module

`mio_client.core.root_manager` module

`mio_client.core.scheduler` module

`mio_client.core.service` module

`mio_client.core.user` module

`mio_client.core.version` module

## Module contents

`mio_client.schedulers` package

### Submodules

`mio_client.schedulers.sub_process` module

## Module contents

`mio_client.services` package

### Submodules

`mio_client.services.doxygen` module

`mio_client.services.fsoc` module

`mio_client.services.init` module

`mio_client.services.regression` module

`mio_client.services.siarx` module

`mio_client.services.simulation` module

## Module contents

### 8.1.2 Submodules

#### 8.1.3 `mio_client.cli` module

#### 8.1.4 Module contents