Mason

Ben Albrecht (Cray Inc.), Sam Partee (Haverford College), Ben Harshbarger, and Preston Sahabu (Cray Inc.)

CHIUW 2018
May 25, 2018
Mason: Motivation

- Previously, modules had to be checked into repository
  - Developers had to sign a CLA
  - Code had to be under a compatible license
  - Code needed to be reviewed by core team

- Modules were gated for release alongside the compiler

- This hinders the ability for users to contribute/share code
Mason: Overview

- **Mason is a package manager and build tool for Chapel**
  - "a skilled worker who builds by laying units of substantial material"
  - Influenced by Rust’s Cargo
  - Basic functionality (version 0.1.0) introduced in Chapel 1.16

- **Written entirely in Chapel**
  - An instance of eating our own dog food.
Mason: Overview

● **Command line tool:** ‘mason’
  ● Builds, runs, and documents packages

● **Centralized registry, decentralized packages,**
  ● Packages exist as TOML files in a single repository
  ● Source code exists somewhere else, like a GitHub repository

● **Dependencies are managed on a per project basis**
  ● Dependency resolution uses semantic versioning
Mason: Outline

- **Basic Usage**
  - Building Mason
  - Creating, Building, and Running a Project
  - Building Documentation
  - Searching for Packages
  - Adding Dependencies
  - Dependency Resolution

- **Mason Registry**

- **Publishing Packages**

- **Planned Features**
Mason: Building Mason

- Mason comes with Chapel release and git repository

- Build mason with ‘make mason’ from $CHPL_HOME
  - Will build Chapel compiler if not already built
  - Symbolically links executable to same directory as ‘chpl’
  - Also supports the ‘make install’ target

```bash
> git clone git@github.com:chapel-lang/chapel.git
> cd chapel
> make mason
```
Mason: Creating a Project

- Create a project with ‘mason new <project name>’

  > mason new MyPackage
  Created new library project: MyPackage

- Initializes an empty git repository

  MyPackage/
  Mason.toml
  src/
  MyPackage.chpl
  .git/
Mason: Creating a Project

- A default manifest, "Mason.toml", is created

  ```toml
  [brick]
  name = "MyPackage"
  version = "0.1.0"
  chplVersion = "1.16.0"

  [dependencies]
  ```

  Packages start as v0.1.0
  Compatible with 1.16 or later
  Zero dependencies

- A default source file is also generated

  ```
  /* Documentation for MyPackage */
  module MyPackage {
    writeln("New library: MyPackage");
  }
  ```
Compile your project with ‘mason build’:

1. Refreshes the registry

2. Creates a lock file, "Mason.lock", also in TOML format
   - Ensures repeatable builds by locking in versions and configurations
     ```bash
     > cat MyPackage/Mason.lock
     [root]
     name = "MyPackage"
     version = "0.1.0"
     chplVersion = "1.16.0..1.16.0"
     ```

3. Downloads dependencies to $MASON_HOME
   - Defaults to $HOME/.mason/

4. Compiles the program into MyPackage/target/debug/
Mason: Running a Project

- Use ‘mason run’ to execute your project

  > mason run
  New library: MyPackage

- Final directory hierarchy:

  MyPackage/
  Mason.toml
  Mason.lock
  src/
  MyPackage.chpl
  target/
  debug/
  myPackage
  .git/
  .git/
Mason: Building Documentation

- Use ‘mason doc’ to build documentation with chpldoc

  > mason doc
  chpldoc src/MyPackage.chpl

- HTML documentation built in MyPackage/docs/

  ![HTML documentation preview](image-url)
Mason: Searching for packages

- **Search with ‘mason search <query>’**
  - Case-insensitive substring matching
  - Lists latest version of packages
  - Empty query will list all packages

```
> mason search E
Alice (0.3.0)
Eve (1.3.0)
MyPackage (0.1.0)

> mason search bo
Bob (1.1.0)
```
Mason: Adding Dependencies

● Add dependencies by modifying Mason.toml
  ● List module dependencies and versions
    ...
    [dependencies]
    Bob = "1.1.0"
    Alice = "0.3.0"

● The next ‘mason build’ will:
  ● Resolve versions and download dependencies to $MASON_HOME
  ● Build the program with the modules in the compiler's module path
    > mason build
    Updating mason-registry
    Downloading dependency: Bob-1.1.0
    Downloading dependency: Alice-0.3.0
Mason: Lock File

- Lock file stores versions and source locations

```python
[root]
name = "MyPackage"
version = "0.1.0"
chplVersion = "1.16.0 .. 1.16.0"
dependencies = ["Bob 1.1.0 https://github.com/BobDev/Bob", ..]

[Bob]
name = "Bob"
version = "1.1.0"
chplVersion = "1.16.0 .. 1.16.0"
source = "https://github.com/BobDev/Bob"
dependencies = [..]

[Alice]
...
```
Mason: Dependency Resolution

● What if there are two versions of a package?

● IVRS relies on semantic versioning
  ● "Incompatible Version Resolution Strategy"
  ● Semantic versioning:
    ● Distinct major versions are incompatible
    ● Use the latest minor version
    ● Use the latest bug fix
Mason: Dependency Resolution

- What if there are two versions of a package?
- IVRS relies on semantic versioning
  - "Incompatible Version Resolution Strategy"
  - Semantic versioning:
    - Distinct major versions are incompatible
    - Use the latest minor version
    - Use the latest bug fix

<table>
<thead>
<tr>
<th>Bob</th>
<th>Alice</th>
<th>Result (Eve)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0.1</td>
<td>1.0.0</td>
<td>1.0.1</td>
</tr>
</tbody>
</table>
Mason: Dependency Resolution

- What if there are two versions of a package?
- IVRS relies on semantic versioning
  - "Incompatible Version Resolution Strategy"
  - Semantic versioning:
    - Distinct major versions are incompatible
    - Use the latest minor version
    - Use the latest bug fix

<table>
<thead>
<tr>
<th>Bob</th>
<th>Alice</th>
<th>Result (Eve)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0.1</td>
<td>1.0.0</td>
<td>1.0.1</td>
</tr>
<tr>
<td>1.2.4</td>
<td>1.3.0</td>
<td>1.3.0</td>
</tr>
</tbody>
</table>
● What if there are two versions of a package?

● IVRS relies on semantic versioning
   ● "Incompatible Version Resolution Strategy"
   ● Semantic versioning:
     ● Distinct major versions are incompatible
     ● Use the latest minor version
     ● Use the latest bug fix

<table>
<thead>
<tr>
<th>Bob</th>
<th>Alice</th>
<th>Result (Eve)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0.1</td>
<td>1.0.0</td>
<td>1.0.1</td>
</tr>
<tr>
<td>1.2.4</td>
<td>1.3.0</td>
<td>1.3.0</td>
</tr>
<tr>
<td>1.0.0</td>
<td>2.0.0</td>
<td>Error</td>
</tr>
</tbody>
</table>
Mason: The Registry

● Mason uses a centralized registry

● Packages are defined by manifest files:
  ```
  mason-registry/
  Bricks/
  Bob/
  1.1.0.toml
  Alice/
  0.3.0.toml
  Eve/
  1.2.4.toml
  1.3.0.toml
  ```

● Registry manifest include an additional ‘source’ field:
  ```
  source = "https://github.com/chapel-lang/MyPackage"
  ```
Mason: The Registry

- Mason can be configured to look elsewhere for registry
  - MASON_REGISTRY – a registry in the form of a git URL
  - Registries can be local git repositories
  - Registries can include local or private git repositories as packages

  MASON_REGISTRY = https://github.com/someUser/custom-registry

- Mason can support multiple registries
  - MASON_REGISTRY can contain comma-separated registries
  - Packages are searched in left-to-right order of MASON_REGISTRY

  MASON_REGISTRY = \\
  "my/local/private/registry, \\
  https://github.com/someUser/custom-registry, \\
  https://github.com/chapel-lang/mason-registry"
Mason: The Registry

● ‘mason env’ lists relevant environment variables
  ● Similar to ‘printchplenv’

> export MASON_REGISTRY=/path/to/shared/registry
> mason env
MASON_HOME: /users/eve/.mason
MASON_REGISTRY: /path/to/shared/registry *
Mason: Publishing a Package to Registry

- **Add git tag to package repository in format of `vX.Y.Z`**
  
  ```bash
  git tag -a v0.1.0 -m "MyPackage 0.1.0"
  ```

- **Fork the mason-registry**

- **Add manifest file to `<package>/<version>.toml`**
  
  - Include additional `source` field

    ```toml
    [brick]
    name = "MyPackage"
    version = "0.1.0"
    chplVersion = "1.16"
    author = "Chapel Lang"
    source = "https://github.com/chapel-lang/MyPackage"
    
    [dependencies]
    ```

- **Open a Pull Request against chapel-lang-lang/mason-registry**
Mason: Planned Features

- **Add support for testing**
  > mason test

- **Simplify publishing of new packages**
  > mason publish

- **Add support for non-Chapel dependencies**

- **Add CI testing for the package ecosystem**

- **And much much more…**
  - See issue #7106 for mason wish list