New Tools and Tool Improvements

Chapel Team, Cray Inc.

Chapel version 1.16

October 5, 2017

Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

Outline

- The ‘mason’ Package Manager
- c2chapel
- chpldoc Improvements

The 'mason' Package Manager

Mason: Background

- **Contributed modules have been bundled into the release**
  - ‘Package’ modules, not part of standard library

- **Bundling such modules with releases is unsustainable**
  - Developers need to sign a CLA to contribute
  - Such modules need to be compatible with Chapel’s license
  - Core team needs to review each module
  - Availability gated by compiler release timing
  - These concerns inhibit healthy community growth

- **A package manager helps solve these problems**
  - Package development/release cycles distinct from compiler
  - Offloads CLA/license requirements to package authors
  - Provides a platform to grow the Chapel ecosystem

Mason: This Effort

- **Designed and implemented a package manager: mason**
  - "a skilled worker who builds by laying units of substantial material"
  - Heavily influenced by Rust’s Cargo
  - Version 0.1.0 - very basic functionality

- **Written entirely in Chapel**
  - An instance of eating our own dog food
    ... to help expose usability issues.
    ... to motivate stabilization of language and APIs.

Mason: This Effort

- **Command line tool: ‘mason’**
  - Builds, runs, and documents packages
- **Decentralized packages, centralized registry**
  - Source code exists somewhere else, like a GitHub repository
  - Packages exist as TOML files in a single repository
- **Dependencies are specified and downloaded per project**
  - Dependency resolution uses semantic versioning

Mason: Basic Usage

- Creating a Project
- Building
- Running
- Adding Dependencies

Mason: Creating a Project

- **Build mason with ‘make mason’ from $CHPL_HOME**
  - Symbolically links executable to same directory as ‘chpl’

- **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
```

Mason: Creating a Project

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

```
[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");
}
```

Mason: Building

- **Compile your project with ‘mason build’**
- **Downloads registry and dependencies to $MASON_HOME**
  - Defaults to $HOME/.mason/
- **Creates a lock file, "Mason.lock", also in TOML format**
  - Ensures repeatable builds by locking in versions and configurations
  - ‘mason update’ - only update/generate the lock file

```
> cat MyPackage/Mason.lock
[root]
name = "MyPackage"
version = "0.1.0"
chplVersion = "1.16.0..1.16.0"
```

Mason: Running

- 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
```

Mason: Adding Dependencies

- **Add dependencies by modifying Mason.toml**
  - List module dependencies and versions

```
...
[dependencies]
Bob = "1.1.0"
Alice = "0.3.0"
```

  - Add 'use' statements to your project

```
use Bob, Alice;
```

- **Dependencies downloaded in next 'mason build'**
  - Mason will:
    - download modules to satisfy dependencies
    - put 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: Adding Dependencies

- Lock file stores versions and source locations

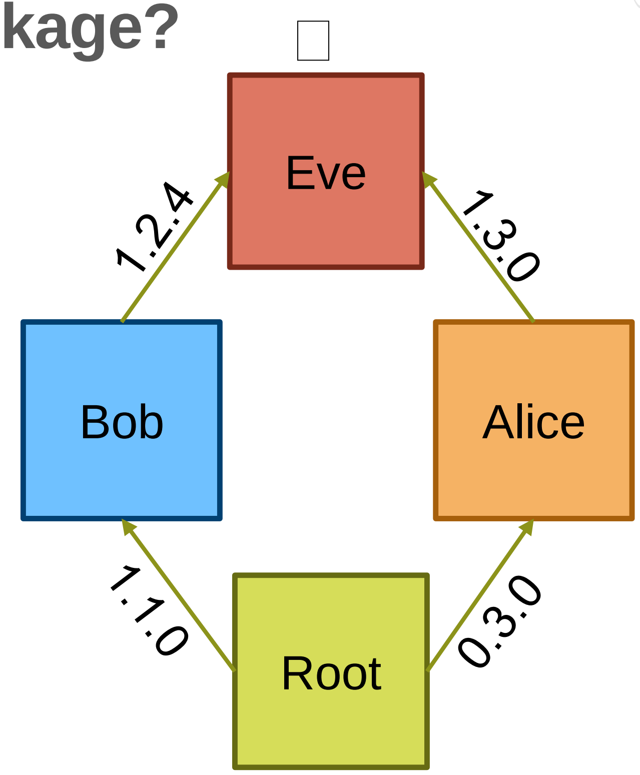
```
[root]
name = "MyPackage"
version = "0.1.0"
chplVersion = "1.16.0 .. 1.16.0"
dependencies = [...]

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

[Alice]
...
```

Mason: Adding Dependencies

- **What if there are two versions of a package?**
- **Dependencies computed with IVRS**
  - "Incompatible Version Resolution Strategy"
  - Follows semantic versioning
    - Distinct major versions are incompatible
    - Use the latest minor version
    - Use the latest bug fix
  - Single version stored in lock file



Bob	Alice	Result (Eve)
1.0.0	2.0.0	Error
1.2.4	1.3.0	1.3.0
1.0.1	1.0.0	1.0.1

Mason: The Registry

- **Mason uses a centralized registry**
  - github.com/chapel-lang/mason-registry

- **Packages are defined by TOML files**

```
mason-registry/
  Bricks/
    Bob/
      1.1.0.toml
    Alice/
      0.3.0.toml
    Eve/
      1.2.4.toml
      1.3.0.toml
```

Mason: The Registry

- **Open a Pull Request to add your package**
  - Ensure source repo has ‘vX.Y.Z’ git tag, v0.1.0 in this example
  - Add a TOML file named after a version: MyPackage/0.1.0.toml

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

[dependencies]
```

Mason: The Registry

- **Search with ‘mason search <query>’**
  - Case-insensitive substring matches
  - Lists latest version of package
  - 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: The Registry

- **Mason can be configured to look elsewhere for registry**
  - MASON_REGISTRY – must be a valid git URL
    - Includes file paths – useful for offline environments
  - ‘mason env’, like ‘printchplenv’, lists relevant environment variables

```
> export MASON_REGISTRY=/path/to/shared/registry
> mason env
MASON_HOME: /users/eve/.mason
MASON_REGISTRY: /path/to/shared/registry *
```

Mason: Status

- **Included in 1.16 release**
- **Version 0.1.0 available for users to try out**
- **Some packages by Chris Taylor available on registry:**
  - MatrixMarket
  - LinearAlgebraJama

Mason : Next Steps

- **Testing and Deployment**
  - Introduce "Blessed" packages to be tested nightly
- **Security**
  - Verify package author identity
- **Improving offline experience**
  - Add commands for caching subset of packages locally
- **Managing C dependencies**
  - Use a backend C dependency manager to support C dependencies
- **Centralized package system**
  - Cache packages themselves, similar to crates.io
- **Next steps tracked in #7106**

c2chapel

c2chapel: Background

- **Chapel supports interoperability with C:**

```
extern proc printf(fmt : c_string, args ...) : void;

extern record myRecord {
  var data : c_ptr(int);
  var len  : c_int;
}
```

- **This is a tedious process for nontrivial C libraries**
  - SQLite, LAPACK, BLAS

- **Users should have a tool to help automate wrapping**

c2chapel: Background

- **Nikhil Padmanabhan (Yale) contributed ‘c2chapel’ script**
  - Python script leveraging ‘pycparser’ package
  - Handled simple function declarations

- **Spent a long time as a second-class utility**
  - Not included in release, only available on master
  - Lots of errors for common C features
  - No regular testing

c2chapel: This Effort

- **Expanded supported C features**
  - Restricted to C99 standard
- **Improved build process and testing**
- **Included in release**

c2chapel: This Effort - Functionality

- **Basic usage:**
  - Accepts C99 header as argument
  - Prints Chapel wrapper to stdout

```
> c2chapel foo.h
```

C99

```
// foo.h
struct misc {
  char a;
  char* b;
  void* c;
  int* d;
};
```

Chapel

```
// Generated with c2chapel version 0.1.0

// Header given to c2chapel:
require "foo.h";

// Note: Generated with fake std headers
extern record misc {
  var a : c_char;
  var b : c_string;
  var c : c_void_ptr;
  var d : c_ptr(c_int);
}
```

c2chapel: This Effort - Functionality

- **Expanded supported C features**
  - function pointers
  - structs with fields
  - typedefs
  - varargs
  - global enums

- **Better handling of standard headers**
  - Original script would break on things like “#include <stdlib.h>”
  - Often related to GNU extensions
  - ‘pycparser’ leverages ‘fake’ headers that redefine tricky constructs

```
typedef int __gnu_va_list;
```

c2chapel: This Effort - Functionality

- Example translations:

C99

```
struct allInts {
  int a;
  unsigned int b;
  long long c;
};


void msg(const char* fmt);
```

Chapel

```
extern record allInts {
  var a : c_int;
  var b : c_uint;
  var c : c_longlong;
}


extern proc msg(fmt : c_string) : void;
```

- More examples in $CHPL_HOME/tools/c2chapel/test/

c2chapel: This Effort - Build/Test

- **Built with 'make c2chapel' from $CHPL_HOME**
  - Placed in same directory as 'chpl' to be $PATH-visible
- **c2chapel installs 'pycparser' to a local virtualenv**
  - Leaves user's python environment untouched
  - Requires an internet connection during 'make'
- **'make check' from $CHPL_HOME/tools/c2chapel**
  - Runs various correctness tests
- **Now tested nightly on master**

c2chapel: Status and Next Steps

Status: c2chapel 0.1.0 included in the 1.16 release

- Significantly more capable and flexible
- Now easily available to users
- Can wrap SQLite, but not LAPACK/BLAS

Next Steps: Improve versatility and installation process

- Improve handling of GNU extensions
- ‘ref’ vs. c_ptr for formals
- Continue to expand testing for other C constructs
- Allow offline installation

chpldoc Improvements

chpldoc Improvements

Background: chpldoc is Chapel's code documentation tool

- Some known bugs remain, but otherwise stable

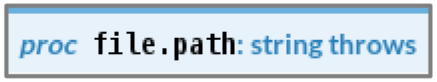
This Effort:

- Documented 'throwing' functions as such
- Added warning when end of doc comment doesn't match the start
  - Contributed by Krishna Keshav
- Added support for math equations in documentation via LaTeX

```
.. math:: a^2 + b^2
```

$\longrightarrow \boxed{a^2 + b^2}$

Impact: Cleaner/more accurate documentation



proc file.path: string throws

Next steps:

- Create a 'throws' section in documentation for throwing procedures
  - list possible errors the routine could throw
- Fix remaining known bugs, respond to user requests

Legal Disclaimer

CRAY CHAPEL

CRAY

THE SUPERCOMPUTER COMPANY