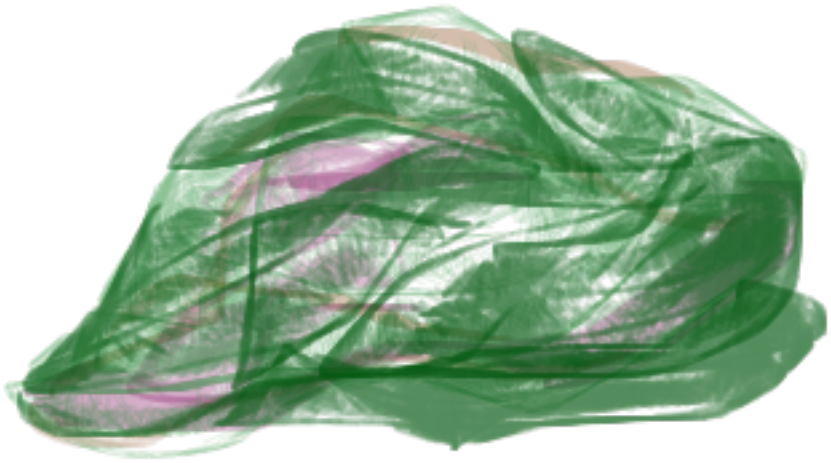
moldspec Documentation

Release 0.1

Matt Haggard

November 02, 2012

CONTENTS



Every configuration management system does things a little differently, which can be frustrating when you want to migrate from one system to another. To that end, this specification exists in hopes of standardizing the components of a configuration management system.

This and the linked documents, known collectively as the *Mold Standard*, define a standard for configuration management systems.

CHAPTER

ONE

DESIGN

1.1 Components

The *Mold Standard* defines

1. The **actors** that use and produce
2. structured **documents**
3. in a pattern of **routines**
4. to make **resources** conform to a desired state.

CHAPTER

TWO

ACTORS

2.1 Fact Checker

The Fact Checker is responsible for getting the facts about a system. It produces a *Fact Document*.

2.2 Prescriber

XXX to be defined

2.3 Inspector

The Inspector inspects the current state of resources.

It accepts an *Identity Document* and returns an *Observation Document* conforming to the schema for the given resource type.

For example, to inspect the state of the file `/tmp/foo` pass this document to the appropriate inspector:

```
{
    "kind": "file",
    "name": "/tmp/foo"
}
```

The inspector will return a document conforming to the *file Observation Schema*:

```
{
    "kind": "file",
    "name": "/tmp/foo",
    "exists": true,
    "size": 3493,
    "sha": "c30a7f7531c41ec102fb5510d58166b502f68437",
    "user": "foo",
    "group": "bar",
    ...
}
```

2.4 Choreographer

Takes the facts, prescription, observed state and lays out the steps.

2.5 Performer

The Performer makes necessary changes to a machine in order to conform to a prescribed state.

It accepts a *Prescription Document* and XXX what it returns is currently undefined.

For example, you might give it this prescription to ensure that the file at `/tmp/foo` exists and has attributes described:

```
{
    "kind": "file",
    "name": "/tmp/foo",
    "exists": true,
    "user": "jim",
    "group": "jimsgroup",
    "content": "This is the content of the file"
}
```

2.6 Historian

Collects logs and things.

CHAPTER

THREE

DOCUMENTS

All documents are in the JSON format. Schema definitions use the JSON Schema format.

3.1 Fact Document

Fact documents describe the (relatively) immutable characteristics of a system. They are returned by *Fact Checkers*.

3.1.1 Schema

os string

Name of the operating system

JSON Schema:

```
{
    "type": "object",
    "properties": {
        "os": {
            "type": "string",
            "description": "Name of the operating system"
        }
    }
}
```

3.1.2 Example

```
{
    'os': {
        'kind': 'linux',
        'distro': 'ubuntu',
        'version': '12.10',
    },
}
```

3.2 Identity Document

Identity documents uniquely identify a resource.

3.2.1 Schema

kind string

Name of resource class

name string

Unique key identifying exactly one resource within a class

JSON Schema:

```
{
    "type": "object",
    "properties": {
        "kind": {
            "type": "string",
            "description": "Name of resource class"
        },
        "name": {
            "type": "string",
            "description": "Unique key identifying exactly one resource within a class"
        }
    }
}
```

3.2.2 Example

For example, the identity document for the `/etc/hosts` file looks like this:

```
{
    "kind": "file",
    "name": "/etc/hosts"
}
```

3.3 Prescription Document

Prescription documents describe the **desired** state of a resource. Each resource type has its own schema for its Prescription documents. Go to *Resources* to see the complete list of resource-specific Prescription documents.

These documents are given to a *Performer* which make the changes necessary to match the prescription.

For example, if we want to make sure the file `/tmp/foo` does not exist, we could prescribe that with this document:

```
{
    "kind": "file",
    "name": "/tmp/foo",
    "exists": false
}
```

XXX The prescription is actually a list of Prescriptions. Somehow, each one should be associated with which steps are a result of it and should be cast into buckets depending on step success/failure (note from notebook – may overlap with steps document).

3.4 Observation Document

Observation documents describe the **actual** state of a resource. Each resource type has its own schema for its *Observation documents*. Go to *Resources* to see the complete list of resource-specific Observation documents.

Inspectors return Observation documents.

For example, a `file` resource observation document might look like this:

```
{
    "kind": "file",
    "name": "/tmp/foo",
    "exists": true,
    "size": 3493,
    "sha": "c30a7f7531c41ec102fb5510d58166b502f68437",
    "user": "foo",
    "group": "bar",
    ...
}
```

3.5 Steps Document

Steps documents contain the steps a Performer needs to follow to bring about the desired state.

Steps documents reference

XXX include schema

CHAPTER

FOUR

ROUTINES

4.1 Mold

This routine finds out what state a system should be in, then does what's needed to make it conform to that state.

Here's some pseudo code describing how the state of a machine is set:

```
facts = FactChecker()
prescription = Prescriber(facts)
while 1:
    observation = Observer(prescription)
    steps = Choreographer(facts, prescription, observation)
    if not steps:
        break
    Performer(steps)
```

An implementation might replace some or most of those function calls with calls to remote systems. Neither error handling nor logging are shown in the pseudo code.

CHAPTER

FIVE

RESOURCES

5.1 `file`

5.1.1 `file` Identity Schema

`kind` (required) `string` matching `"file"`

Indicates that this is a file resource

`name` (required) `string`

Absolute path of file

JSON Schema:

```
{
    "type": "object",
    "properties": {
        "kind": {
            "pattern": "file",
            "required": true,
            "type": "string",
            "description": "Indicates that this is a file resource"
        },
        "name": {
            "required": true,
            "type": "string",
            "description": "Absolute path of file"
        }
    }
}
```

5.1.2 `file` Observation Schema

`exists` (required) `boolean`

`true` if the file exists, `false` if it doesn't

`kind` (required) `string` matching `"file"`

Indicates that this is a file resource

`name` (required) `string`

Absolute path of file

group string

Name of the group owning the file

owner string

Name of the user owning the file

permissions integer

Octal permission bits for the file, e.g. `0755`. Since it's an integer you will need to convert to octal if you want it in that format.

sha string

SHA1 hash of the file's contents as a hexadecimal string

size ['integer', 'long']

Current file size in bytes

JSON Schema:

```
{
    "type": "object",
    "properties": {
        "kind": {
            "pattern": "file",
            "required": true,
            "type": "string",
            "description": "Indicates that this is a file resource"
        },
        "group": {
            "type": "string",
            "description": "Name of the group owning the file"
        },
        "name": {
            "required": true,
            "type": "string",
            "description": "Absolute path of file"
        },
        "exists": {
            "required": true,
            "type": "boolean",
            "description": "``true`` if the file exists, ``false`` if it doesn't"
        },
        "sha": {
            "type": "string",
            "description": "SHA1 hash of the file's contents as a hexadecimal string"
        },
        "owner": {
            "type": "string",
            "description": "Name of the user owning the file"
        },
        "permissions": {
            "type": "integer",
            "description": "Octal permission bits for the file, e.g. ``0755``.  Since it's an integer
        },
        "size": {
            "type": [
                "integer",
                "long"
```

```
            ],
            "description": "Current file size in bytes"
        }
    }
}
```

5.1.3 `file` Prescription Schema

`content` (required) `string`

URI or local absolute path to the file

`exists` (required) `boolean`

`true` if the file should exist, `false` if it should not exist

`kind` (required) `string` matching `"file"`

Indicates that this is a file resource

`name` (required) `string`

Absolute path of file

`group` `string`

Name of the group owning the file

`owner` `string`

Name of the user owning the file

`permissions` `integer`

Octal permission bits for the file, e.g. `0755`. Since it's an integer you will need to convert to octal if you want it in that format.

JSON Schema:

```
{
    "type": "object",
    "properties": {
        "content": {
            "required": true,
            "type": "string",
            "description": "URI or local absolute path to the file"
        },
        "kind": {
            "pattern": "file",
            "required": true,
            "type": "string",
            "description": "Indicates that this is a file resource"
        },
        "group": {
            "type": "string",
            "description": "Name of the group owning the file"
        },
        "name": {
            "required": true,
            "type": "string",
            "description": "Absolute path of file"
        },
```

```
        "exists": {
            "required": true,
            "type": "boolean",
            "description": "``true`` if the file should exist, ``false`` if it should not exist"
        },
        "owner": {
            "type": "string",
            "description": "Name of the user owning the file"
        },
        "permissions": {
            "type": "integer",
            "description": "Octal permission bits for the file, e.g. ``0755``.  Since it's an integer
        }
    }
}
```

5.2 `user`

5.2.1 `user` Identity Schema

`kind` (required) `string` matching `"user"`

Indicates that this is a user resource

`name` (required) `string`

Name of user

JSON Schema:

```
{
    "type": "object",
    "properties": {
        "kind": {
            "pattern": "user",
            "required": true,
            "type": "string",
            "description": "Indicates that this is a user resource"
        },
        "name": {
            "required": true,
            "type": "string",
            "description": "Name of user"
        }
    }
}
```

5.2.2 `user` Observation Schema

`exists` (required) `boolean`

true if the user exists; false if it doesn't

`kind` (required) `string` matching `"user"`

Indicates that this is a user resource

name (required) `string`

Name of user

comment `string`

A description of the user

gid `string`

User's group ID or name

home `string`

Path to user's home

password `string`

Password hash

shell `string`

Path to the shell for user

uid `string`

User's user ID

JSON Schema:

```
{
    "type": "object",
    "properties": {
        "comment": {
            "type": "string",
            "description": "A description of the user"
        },
        "kind": {
            "pattern": "user",
            "required": true,
            "type": "string",
            "description": "Indicates that this is a user resource"
        },
        "shell": {
            "type": "string",
            "description": "Path to the shell for user"
        },
        "name": {
            "required": true,
            "type": "string",
            "description": "Name of user"
        },
        "exists": {
            "required": true,
            "type": "boolean",
            "description": "true if the user exists; false if it doesn't"
        },
        "gid": {
            "type": "string",
            "description": "User's group ID or name"
        },
        "home": {
            "type": "string",
            "description": "Path to user's home"
```

```
        },
        "password": {
            "type": "string",
            "description": "Password hash"
        },
        "uid": {
            "type": "string",
            "description": "User's user ID"
        }
    }
}
```

5.2.3 `user` Prescription Schema

`exists` (required) `boolean`

true if the user should exist; false if it should not exist

`kind` (required) `string` matching `"user"`

Indicates that this is a user resource

`name` (required) `string`

Name of user

`comment` `string`

A description of the user

`gid` `string`

User's group ID or name

`home` `string`

Path to user's home

`password` `string`

Password hash

`shell` `string`

Path to the shell for user

`uid` `string`

User's user ID

JSON Schema:

```
{
    "type": "object",
    "properties": {
        "comment": {
            "type": "string",
            "description": "A description of the user"
        },
        "kind": {
            "pattern": "user",
            "required": true,
            "type": "string",
```

```
            "description": "Indicates that this is a user resource"
        },
        "shell": {
            "type": "string",
            "description": "Path to the shell for user"
        },
        "name": {
            "required": true,
            "type": "string",
            "description": "Name of user"
        },
        "exists": {
            "required": true,
            "type": "boolean",
            "description": "true if the user should exist; false if it should not exist"
        },
        "gid": {
            "type": "string",
            "description": "User's group ID or name"
        },
        "home": {
            "type": "string",
            "description": "Path to user's home"
        },
        "password": {
            "type": "string",
            "description": "Password hash"
        },
        "uid": {
            "type": "string",
            "description": "User's user ID"
        }
    }
}
```

CHAPTER

SIX

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*