Pluggable Resolvers

Pluggable resolvers allow you to use resolvers created by 3rd party JavaScript developers — including overriding default resolvers used by Bower.

For example, resolvers can be used for:

- Handling Mercurial or Bazaar repositories
- Speeding up checkouts of services like GitLab or Bitbucket
- Allowing to use packages from npm or component.io
- Proxying downloads through 3rd party service like Artifactory or Nexus Repository
- Implementing custom private registry (hosted on GitHub?)
- Adding authentication support for private GitHub Enterprise instances

Pluggable resolvers were introduced in Bower 1.5. Please make sure your Bower version is correct (`bower --version`).

Using

A Pluggable Resolver is just an npm package that you install as `devDependency` in the `package.json` of your repository, or install globally with `npm install -g`.

Declare what Pluggable resolvers your project uses by adding entries to the `resolvers` section of .bowerrc.

```
{
  "resolvers": [
    "bitbucket-resolver",
    "github-enterprise-resolver"
  ]
}
```

Bower tries to use resolvers in the order specified. If no custom resolver matches the source being processed, Bower fallbacks to default resolvers (git, github, filesystem, svn, registry).

You can find the list of available Bower resolvers on npm website.

Creating

As mentioned, custom resolvers are npm packages with specific a API described below.

The `package.json` should not list `bower` as a `dependency` or `peerDependency` (both have undesired behavior in npm 2.x, and we don't want you to use bower internals). Instead, you can check for proper environment in resolver's factory by reading provided `bower.version` parameter and use any other packages on npm (like request).

Packages should list `bower-resolver` as one of the `keywords` in `package.json`. Resolvers should also follow semver specification.

Here is how an example `package.json` of a custom resolver can look like:

```
{
  "name": "custom-bower-resolver",
  "version": "1.0.0",
  "keywords": ["bower-resolver"],
  "main": "index.js",
  "dependencies": {
```

```
    "request": "^2.61.0"
  }
}
```

The `index.js` should export factory for resolver, as follows:

```
var tmp = require('tmp');

/**
 * Factory function for resolver
 * It is called only one time by
Bower, to instantiate resolver.
 * You can instantiate here any caches
or create helper functions.
 */
module.exports = function resolver
(bower) {

  // Resolver factory returns an
instance of resolver
  return {

    // Match method tells whether
resolver supports given source
    // It can return either boolean or
promise of boolean
    match: function (source) {
      return source.indexOf('svn://')
=== 0
    },

    // Optional:
    // Can resolve or normalize
sources, like:
    // "jquery" =>
"git://github.com/jquery/jquery.git"
    locate: function (source) {
```

```
      return source;
    },

    // Optional:
    // Allows to list available
versions of given source.
    // Bower chooses matching release
and passes it to "fetch"
    releases: function (source) {
      return [
        { target: 'v1.0.0', version:
'1.0.0' },
        { target: 'v1.0.1', version:
'1.0.1' }
      ]
    },

    // It downloads package and
extracts it to temporary directory
    // You can use npm's "tmp" package
to tmp directories
    // See the "Resolver API" section
for details on this method
    fetch: function (endpoint, cached)
{
      // If cached version of package
exists, re-use it
      if (cached && cached.version) {
        return;
      }

      var tempDir = tmp.dirSync();

      // ... download package to
tempDir

      return {
```

```
            tempPath: tempDir.name,
            removeIgnores: true
          }
        }
      }
    }
```

If you need something more solid, see this real world example: Mercurial Resolver.

Resolver API

Resolver package

```
var plugResolver = require('pluggable-
resolver')

var resolver = plugResolver({
  version: '1.5.0',
  config: {...},
  logger: logger
})
```

- `resolver`: `Resolver` - instance of the resolver.
- `version`: `String` - Bower's version that instantiates resolver. You can validate it.
- `config`: `Object` - Bower's config. You can ask authors to put extra configuration in it.
- `logger`: `Object` - Bower's logger. Use it to output important warnings / information.

`plugResolver()` returns an instance of the resolver with the API described below.

```
resolver.match()
resolver.locate()
resolver.releases()
resolver.fetch()
```

resolver.match()

Tells Bower whether to use or not use this resolver for some source.

```
var isMatched = resolver.match( source
)
```

- `source`: `String` - source from bower.json, like `git://github.com/jquery/jquery.git`
- `isMatched`: `Boolean` - *Returns* a boolean that tells whether resolver can handle given source (either by locating them with `locate` method, or fetching it with `fetch` + optional `releases` method).

`.match()` can also return a Promise of the result. It's useful e.g. for filesystem checks.

resolver.locate()

Allows to implement simplified registry.

```
var locatedSource = resolver.locate(
source )
```

- `source`: `String` - source from bower.json, like `jquery/jquery`
- `locatedSource`: `String` - *Returns* a resolved source string, like `"git://github.com/jquery/jquery.git"`

`.locate()` can also return a Promise of the result. It's useful e.g. for remote registry calls.

resolver.releases()

Bower selects one matching `version` from the result and passes matching `target` field to `fetch` method.

```
var resolvedReleases =
resolver.releases( source )
```

- `source`: `String` - source from bower.json, like `git://github.com/jquery/jquery.git`
- `resolvedReleases`: `Array` - *Returns* available releases for given source (like list of available tags on GitHub)
  - `target`: `String` - unique target id for release (usually tag name)
  - `version`: `String` - semantic version for the target above

`.releases()` can also return a Promise of the result.

resolver.fetch()

Downloads given endpoint and returns path to temporary directory.

```
var fetched = resolver.fetch(
endPoint, cached )
```

- `endpoint`: `Object` - endpoint for the resource to download
  - `name`: `String` - name of resource (like `jquery`)
  - `source`: `String` - where to download resource from (like `git://github.com/jquery/jquery.git`)
  - `target`: `String` - the version or release of resource to download (like `v1.0.0`)
- `cached`: `Object` - contains information about cached resource
  - `endpoint`: `Object` - endpoint of cached resource (the same format as above)
  - `release`: `String` - release of cached resource
  - `releases`: `Array` - the result of `releases` method
  - `version`: `String` - present cached resource has been resolved as version (like `1.0.0`)

    - `resolution: String` - the "resolution" returned from previous fetch call for same resource
- `fetched: Object` - *Returned*
    - `tempPath: String` - path to teporary directory with downloaded resource
    - `removeIgnores: Boolean` - tells whether bower should remove files ignores in bower.json.
    - `resolution: Object` - extra object that is saved in `.bower.json` and passed in `cached` field to the next `fetch` call. It can be used e.g. to download resources conditionally, for example by storing e-tag or last-modified time.

`.fetch()` can also return a Promise of the result.

If `.fetch()` returns `undefined`, then Bower re-uses cached package.

Help improve these docs. Open an issue or pull request.