StrongLoop
An IBM Company

StrongLoop Reference Documentation

Frequently asked questions

General questions

StrongLoop supports the following operating systems:

- Red Hat Enterprise Linux (RHEL)/CentOS 6.3 (RPM)
- RHEL 6 and RHEL 7 on IBM z systems
- Debian/Ubuntu 12.10 (DEB)
- SUSE Linux Enterprise Server 11 and 12 on IBM z systems (with Node 0.12.7 and 4.2.2)
- Mac OS X Mountain Lion 10.8 (PKG)
- Microsoft Windows 8, 2008 (MSI)
  NOTE: Node does not support using Cygwin. Instead use Windows Command Prompt for command-line tools.

What version of Node do you support?

We support most recent versions of Node, but for best results use the latest stable version of Node, as stated on http://nodejs.org/.

What is slc?

`slc` is a command line tool for building and managing applications. Use it to:

- Rapidly create LoopBack applications and REST APIs, using `slc loopback`. See Creating an application.
- Build and deploy any Node application and manage with StrongLoop Process Manager.
- Run an application in place under control of StrongLoop PM, using `slc start`. See slc start for more information.
- Update applications with zero downtime.

For a complete reference, see Command-line reference.

How do I install the latest version of StrongLoop?

In brief, just use:

```
$ npm install -g strongloop
```

For more information, see Installing StrongLoop.

See Updating to the latest version to update to the latest version.

LoopBack

See LoopBack FAQ for answers to additional detailed LoopBack questions.

Is LoopBack free? How much does it cost?

There are free and paid versions of LoopBack. See http://strongloop.com/node-js/subscription-plans/ for more information.

LoopBack uses a dual license model: you may use it under the terms of the open source MIT license, or under the commercial StrongLoop License. See the license file for links to both licenses.

Is there a developer forum or mailing list?

Yes! The LoopBack Google Group is a place for devlopers to ask questions and discuss LoopBack and how they are using it. Check it out!

There is also a LoopBack Gitter channel for realtime discussions with fellow LoopBack developers.

StrongLoop also publishes a blog with topics relevant to LoopBack; see Blog posts for a list of the latest posts.

What client SDKs does LoopBack have?

LoopBack has three client SDKs for accessing the REST API services generated by the LoopBack framework:

- iOS SDK (Objective C) for iPhone and iPad apps. See iOS SDK for more information.
- Android SDK (Java) for Android apps. See Android SDK for more information.
- AngularJS (JavaScript) for HTML5 front-ends. See AngularJS JavaScript SDK for more information.

Which data connectors does LoopBack have?

LoopBack provides numerous connectors to access enterprise and other backend data systems.

Database connectors:

- Cloudant connector
- DB2 connector
- Memory connector
- MongoDB connector
- MySQL connector
- Oracle connector
- PostgreSQL connector
- Redis connector
- SQL Server connector

Other connectors:

- Email connector
- Push connector
- Remote connector
- REST connector
- SOAP connector
- Storage connector

Additionally, there are community connectors created by developers in the LoopBack open source community.

Why do curl requests to my LoopBack app fail?

If the URL loads fine in a browser, but when you make a `curl` request to your app you get the error:

```
curl: (7) Failed to connect to localhost port 3000: Connection refused
```

The cause is likely to be because of incompatible IP versions between your app and `curl`.

On Mac OS 10.10 (Yosemite), `curl` uses IP v6 by default.

LoopBack, by default uses IP v4, and `curl` might be using IP v6. If you see IP v6 entries in your hosts file (::1 localhost, fe80::1%lo0 localhost), it is likely that `curl` is making requests using IP v6. To make request using IP v4, specify the `--ipv4` option in your curl request as shown below.

```
$ curl http://localhost:3000 --ipv4
```

You can make your LoopBack app use IP v6 by specifying an IP v6 address as shown below:

```
app.start = function() {
  // start the web server
  return app.listen(3000, '::1',function() {
    app.emit('started');
    console.log('Web server listening at: %s', app.get('url'));
  });
};
```

If you are trying query filters with curl, use the `-g` or `--globoff` option to use brackets `[` and `]` in request URLs.

Application monitoring

What endpoints can you monitor with StrongLoop?

The endpoints you can monitor include:

- HTTP requests – URLs and parameters
- Memory object caching – Memcached
- Web-services – SOAP, REST, HTTP
- Relational databases - MySQL, Oracle
- Including code to connect to MongoDB backends
- NoSQL data stores like MongoDB

Is it possible to get a heap dump when I notice an issue?

You can get detailed heap snapshots with StrongLoops tools. See Taking heap snapshots for details.

What is CPU profiling?

StrongLoop CPU profiling collects the CPU footprint of the application, modules, functions and lines of code. You can start and stop CPU profiling with Arc or `slc`. Once the profile is collected, you can view the following analytics:

- Time distribution of code calls in perspective of CPU footprint over the collection period
- Call chains with insight into multiple child elements
- Ability to hide or explore call chain child elements taking less than 5% time of parent or with single child tiers.

For more information, see Profiling.

Other questions

What is a "private registry?"

Node applications use `npm` to automate the process of installing, upgrading, configuring, and removing Node packages. It automatically handles dependencies, so when you install a package, it will automatically install other packages that it requires. The standard public registry is npmjs.org, which maintains a database of node packages (over 47,000 and growing).

A private registry is a “non-public” source and configuration management system for node projects that enable an enterprise to use only the modules required by their project while being able to share the common registry across multiple teams within their organization. These registries allow node projects to whitelist and manage proliferation of module functionalities, versions, and source.

Once you have a private registry set up, StrongLoop Controller enables you to easily switch between it and the public registry, and to promote packages from private to public registry. For more information, see Using multiple package registries .

Glossary

CONTENTS

A

ACL

Access control list, a list associated with an object that identifies all the subjects that can access the object and their access rights. See Authentication, authorization, and permissions.

API

Application Programming Interface. An interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program.

adapter

Provides the transport-specific mechanisms to make remote objects (and collections thereof) available over their transport. See Strong remoting.

Android

A mobile operating system created by Google, most of which is released under the Apache 2.0 and GPLv2 open source licenses.

AngularJS

Open-source client JavaScript framework.

B

boot script

A JavaScript function that runs automatically when an application starts, by default in the `/server/boot` directory. See Defining boot scripts.

built-in model

One of the predefined models that every LoopBack application has by default. See Using built-in models.

C

cluster

A set of identical Node worker processes all receiving requests on the same port. See also: *worker*.

component

A predefined package that extend a basic LoopBack application. Fundamentally, a component is related code bundled together as a unit to enable LoopBack applications for easy reuse. See LoopBack components.

connector

See *LoopBack connector*.

D

data source

A data source connects with specific database or other back-end system using a connector.   See Connecting models to data sources.

E

endpoint

See *route*.

enterprise connector

A module that connects to back-end data sources such as Oracle, MySQL, or MongoDB.

environment variable

A variable that defines an aspect of the operating environment for a process. For example, environment variables can define the home directory, the command search path, the terminal in use, or the current time zone.

event loop

Single-threaded execution process that runs a Node.js application, typically making a series of asynchronous function calls.

F

flame graph

A visual representation of an application's function call stack and the corresponding execution times. See Tracing.

G

generator

Interactive command-line tool that scaffolds all or part of a LoopBack application. See slc loopback command-line reference.

H

heap snapshot

A data set that shows memory distribution among an application's JavaScript objects at a specific point in time. See Taking heap snapshots.

I

iOS

A closed source, proprietary mobile operating system for Apple devices.

J

JSON

JavaScript Object Notation; a lightweight data-interchange format that is based on the object-literal notation of JavaScript. JSON is programming-language neutral but uses conventions from various languages.

L

LDL

LoopBack Definition Language, a form of JSON used to define LoopBack models and other configurations.

license key

A long character string that enables you to use StrongLoop product features that require a license. See Managing your licenses.

load balancer

Software or hardware that distributes workload across a set of servers to ensure that servers are not overloaded. The load balancer also directs users to another server if the initial server fails.

LoopBack connector

A connector that provides access to a back-end system such as a database, REST API, or other service.

LoopBack DataSource Juggler

An object-relational mapping that provides a common set of interfaces for interacting with databases, REST APIs, and other data sources.

LoopBack model

A model that consists of application data, validation rules, data access capabilities, and business logic that provides a REST API by default.

M

MBaaS

Mobile backend as a service. A computing model that connects mobile applications to cloud computing services and provides features such as user management, push notifications, and integration with social networks through a unified API and SDK.

middleware function

A function executed when an HTTP request is made to a specified REST endpoint. Since LoopBack is based on Express, LoopBack middleware is the same as Express middleware. See Defining middleware.

middleware phase

A stage in application execution when you can call a middleware function. See Defining middleware.

model

See *LoopBack model*.

model property

A value attached to a model; for persisted models, corresponds to a database column or field

N

npm

Node package manager, the command-line tool for installing applications and managing dependencies using the npm registry.

O

on-premises

Pertaining to software that is installed and run on the local computers of a user or organization.

operation hook

Code that is triggered by a model's high-level create, retrieve, update, or delete (CRUD) operations. See Operation hooks.

P

Persisted model

A LoopBack model attached to a persistent data source that automatically gets basic create, read, update, and delete methods.

PaaS

Platform as a service. A cloud computing service that provides a platform that enables customers to develop, run, and manage applications without the building and maintaining the infrastructure typically required to develop and launch an app.

process ID (pid)

The unique identifier that represents a process. A process ID is a positive integer and is not reused until the process lifetime ends.

production

The stage in the software lifecycle when an application or API is generally available to its end-users or consumers. Contrast with "development" and "testing". Also referred to as "deployment".

production host

A server running a production application.

property

See model property.

push notification

An alert or message to a mobile app. See Push notifications.

R

remote object

A JavaScript object exported over the network by a StrongLoop application in the same way you export functions from a module. You can invoke methods on remote objects locally in JavaScript.

REST

A software architectural style for distributed hypermedia systems like the World Wide Web. The term is also often used to describe any simple interface that uses XML (or YAML, JSON, plain text) over HTTP without an additional messaging layer such as SOAP.

route

Part of a URL that identifies a resource. For example, in `http://foo.com/products/id`, the route is `/products/id`.

runtime

Pertaining to the time period during which a computer program is running.

S

SDK

Software development kit. A set of tools, APIs, and documentation to assist with the development of software in a specific computer language or for a particular operating environment.

slc

The StrongLoop command-line tool for development and operations. It provides command line access to all StrongLoop facilities. For more information, see Operating Node applications.

synchronization

The process by which data consistency is achieved between two endpoints such as a provider application and a mobile application. During this process, at either endpoint, data can be updated, created, or deleted. See Synchronization.

W

worker

A Node.js child process.

Command-line reference

Use the `slc` command-line utility to create, manage, and work with LoopBack applications and other Node applications.

Command syntax

```
slc [command] [sub-command] [options] [args]
```

Some commands and sub-commands have *required* "options," as noted in the command reference for the specific command. Such cases are not accurately covered by the above syntax.

Syntax conventions

The documentation uses the following conventions for command-line syntax. In general:

- An item in *`monospace italics`* represents a variable for which you can substitute a string or number, as specified for the specific command.
- An item enclosed in square brackets `[ .. ]` represents an optional item.

Then:

`command` indicates one of the `slc` commands (see table below).

`[`*`sub-command`*`]` indicates a sub-command, for commands that take a sub-command, for example `runctl` or `ctl`.

`[`*`options`*`]` indicates one or more command options, possibly with corresponding values, as described for the particular command. Options, as their name implies, are typically optional. However, some commands or sub-commands have *required* "options," as noted in the command reference for the specific command.

Syntax for options is:

`-x` ***`value`*****:** Single-letter option; separate value (if provided) with a space.
`--extra=value` or **`--extra value`:** Full spelled-out option; separate value (if provided) with a space or an equals sign (=).

Where value is either:

- ***`<foo>`***, a value that you must provide, either a string or number, as described for the particular command.
- **`[`*`foo`*`]`**, a value that you may optionally provide. It is not required.

`[`*`args`*`]` is an optional argument.

Most `slc` commands have the two standard options below. Most commands also have additional options.

STANDARD OPTIONS

`-h`, `--help`
Display help information.

`-v`, `--version`
Display version number.

Commands

The following table summarizes `slc` commands. Each command has specific options and arguments, and some provide sub-commands, as described in each command reference article.

Command	Description
slc arc	Start StrongLoop Arc
build	Build a Node application package, preparing it to be deployed.
debug	Debug module with Node Inspector.
deploy	Deploy a Node application to a process manager.
env	Display information about the Node runtime environment.
loopback	Scaffold a LoopBack application. See Command-line reference (slc loopback) for details.
pm	Manage deployed Node applications with StrongLoop Process Manager.
ctl	Control applications using StrongLoop Process Manager.
pm-install	Install StrongLoop Process Manager as an OS service.
registry	Switch between different npm registries.
start	Run an application locally, under control of StrongLoop Process Manager.

Getting the latest version of slc

See Updating to the latest version for instructions.

slc arc

Run StrongLoop Arc, by default opening it in a web browser window.

SYNOPSIS

```
slc arc [options]
```

OPTIONS

`--cli`
Start the backend only; do not open the browser.

`--licenses`
Start Arc and display the Licenses page. See Managing your licenses for more information.

STANDARD OPTIONS

`-h`, `--help`
Display help information.

`-v`, `--version`
Display version number.

StrongLoop Arc will use a different port number each time you run it. To run it on a specific port, use the PORT environment variable, for example:

```
$ PORT=4000 slc arc
```

slc build

Build a Node application package, preparing it for production. See Installing dependencies for more information on using this command.

SYNOPSIS

```
slc build [options]
```

With no options, the default depends on whether the current app is in a Git repository (if there is a `.git` sub-directory in the current app root directory):

- If in a Git repository, the default option is `--git`, which installs and commits the build results to the "deploy" branch. If a "deploy" branch does not exist, the tool will create it.
- If not, the default options are `--npm` which will bundle, install, and pack the build results into a `<package-name>-<version>.tgz` file.

OPTIONS

`-b, --bundle`
Include dependencies in tar (.tgz) file.

`-c, --commit  [--onto <branch> ]`
Commit build output to specified branch, and create the branch if necessary. Default branch is "deploy." This option requires that Git is installed.

`-i, --install [--scripts]`
Install dependencies, without scripts, by default.
Add the `--scripts` option to run scripts (to build add-ons).

`-g, –git`
Shortcut for `--install --commit`.

`-n, --npm`
Shortcut for `--install --bundle --pack`.

`-p, --pack`

Pack into a publishable archive. Output is a .tgz file in the format produced by `npm pack` and accepted by `npm install`.

STANDARD OPTIONS

`-h, --help`
Display help information.

`-v, --version`
Display version number.

Using Git

See Committing a build to Git for more information.

slc ctl

Control StrongLoop Process Manager. This command enables you to:

- Start, stop, and restart applications that are under management of StrongLoop PM.
- Start and stop application clusters; change number of workers in a cluster.
- Start and stop CPU profiling, take heap snapshots, and perform object tracking.
- Modify environment settings while an application is running.

See also:

- Using Process Manager
- Clustering
- Logging
- Profiling with slc
- Monitoring with slc

This command works only with applications running under control of StrongLoop Process Manager. For local use, that means you must run the application with `slc start`, not `node .` or `slc run` (that don't run applications in Process Manager).

SYNOPSIS

```
slc ctl [options] [sub-command]
```

OPTIONS

`-C, --control <ctl>`
Control endpoint for Process Manager. For a remote Process Manager, this must specify the URL on which the Process Manager is listening.

If Process Manager is using HTTP authentication then you must set valid credentials in the URL, in the form `http://username:password@example.com:7654`.

To tunnel over SSH using an HTTP URL, use the protocol `http+ssh`., for example `http+ssh://example.com:7654`.

- The SSH username defaults to your current user. Override the default with the SSH_USER environment variable.
- Authentication defaults to your current `ssh-agent`. Override the default with the SSH_KEY environment variable specifying the path of an existing private key to use.
- SSH port defaults to 22. Override the default by setting the SSH_PORT environment variable.

Use the STRONGLOOP_PM environment variable to set a default value for the `--control` option; this eliminates the need to supply the option every time.

If you don't specify a channel with this option, the tool uses the following in this order of precedence:

1. STRONGLOOP_PM environment variable, that can specify a local domain path, or an HTTP URL. Use an HTTP URL to specify a remote Process Manager. Use `localhost` for a local Process Manager. The URL must specify at least the process manager's listen port, such as `http://example.com:7654` (default is 8701).
2. `./pmctl:` Process Manager running the current working directory, if any.
3. `~/.strong-pm/pmctl`: Process Manager running in the user's home directory.
4. `/var/lib/strong-pm/pmctl`: Process Manager installed by `slc pm-install`.
5. `http://localhost:8701`: Process Manager running on localhost.

STANDARD OPTIONS

`-h, --help`
Display help information.

`-v, --version`
Display version number.

SUB-COMMANDS

The default sub-command is `status`.

This command has three types of sub-commands:

- Global commands that apply to Process Manager itself.
- Commands that apply to a specific service.
- Commands that apply to a specific worker process.

When you deploy an application to Process Manager, you give the deployed application instance a name, referred to as the *service name* and indicated in command arguments as `<service>`. By default, it is the `name` property from the application's `package.json`.

Process Manager also automatically generates an integer ID for each application it's managing. Typically, the IDs start with one (1) and are incremented with each application deployed; however, the value of ID is not guaranteed. Always determine it with `slc ctl status` once you've deployed an app.

A service becomes available over the network at `http://hostname:port` where:

- `hostname` is the name of the host running Process Manager
- `port` is 3000 + service ID.

For example, if Process Manager is running on my.host.com, then service ID 1 is available at `http://my.host.com:3001`, service ID 2 at `http://my.host.com:3002`, and so on.

Command	Description	Arguments
Global sub-commands		
info	Display information about Process Manager.	
ls	List services under management.	
shutdown	Stop the process manager and all applications under management.	
Service sub-commands (apply to a specific service) The argument <*service*> is the name or ID of a service.		
create <*service*>	Create application instance *service*.	<*service*>, name or ID of the service to create.
cluster-restart <*service*>	Restart the current application cluster workers.	<*service*>, name of target service.
env[-get] <*service*> [*env*...]	List specified environment variables for <*service*>. If none are given, list all variables.	<*service*>, name or ID of target service. <*env*>, one or more environment variables.
env-set <*service*> <*env*>=<*val*>...	Set one or more environment variables for <*service*> and hard restart it with new environment.	<*service*>, name or ID of target service. One or more environment variables, <*env*>, and corresponding value <*val*>.
env-set <*service*> PORT=<*n*>	Run service on the specified port instead of the automatically-generated port. Normally, PM sets the port to a value guaranteed to be different for each app: Use this sub-command to override this behavior. **Do not specify a port already in use**. Doing so will cause the app to crash.	<*service*>, name or ID of target service. <n>, integer port number to use.

env-unset <*service*> <*env*>...	Unset one or more environment variables for <*service*> and hard restart it with the new environment.	<*service*>, name or ID of target service. <*env*>, one or more environment variables.
log-dump <*service*> [--follow]	Empty the log buffer, dumping the contents to stdout. Use `--follow` to continuously dump the log buffer to stdout.	<*service*>, name or ID of target service.
npmls <*service*> [*depth*]	List dependencies of service with id <*service*>	<*service*>, name or ID of target service. *depth*, an integer limit of levels for which to list dependencies; default is no limit.
remove <*service*>	Remove <*service*>.	<*service*>, name or ID of target service.
restart <*service*>	Hard stop the current application: kill the supervisor and its workers with SIGTERM; then restart the current application with new configuration.	<*service*>, name or ID of target service.
set-size <*service*> <*n*>	Set cluster size for <*service*> to <*n*> workers.	<*service*>, name or ID of target service. <*n*>, positive integer.
start <*service*>	Start <*service*>.	<*service*>, name or ID of target service.
status [*service*]	Report status. This is the default command.	*service*, optional name of target service. Default is to show status for all services.
stop <*service*>	Hard stop <*service*>: Kill the supervisor and its workers with SIGTERM.	<*service*>, name or ID of target service.
soft-stop <*service*>	Notify workers they are being disconnected, give them a grace period to close existing connections, then stop the current application.	<*service*>, name or ID of target service.
soft-restart <*service*>	Notify workers they are being disconnected, give them a grace period to close existing connections, then restart the current application with new configuration.	<*service*>, name or ID of target service.
tracing-start <*service*>	Restart all workers with tracing on.	<*service*>, name or ID of target service.
tracing-stop <*service*>	Restart all workers with tracing off.	<*service*>, name or ID of target service.
Worker process sub-commands (apply to a specific worker process) The argument <*worker*> is a worker specification; either: • <*service_id*>.1.<*worker_id*>, where <*service_id*> is the service ID and <*worker_id*> is the worker ID. • <*service_id*>.1.<*process_id*>, where <*service_id*> is the service ID and <*process_id*> is the worker process ID (PID).		
cpu-start <*worker*> [*timeout* [*stalls*]]	Start CPU profiling on worker or process ID <*id*>. Use cpu-stop to save the profile data. NOTE: Requires Node version 0.11 or higher. Saves profiling data to a file you can view with Chrome Dev Tools. See CPU profiling for more information.	<*worker*>, a worker specification (see above). **Linux only**: [*timeout*], timeout period (ms) for Smart profiling. Start CPU profiling when the specified process's Node event loop stalls for more than the specified timeout period. [*stalls*], number of event loop stalls after which the profiler will be stopped automatically (default is 0, never auto-stop). For more information, see Smart profiling with slc.

cpu-stop <*worker*> [*filename*]	Stop CPU profiling on worker process <*id*> and save results in file *`name`*`.cpuprofile`.	<*worker*>, a worker specification (see above). *filename*, optional base file name; default is `node.<`*`id`*`>.cpuprofile`.
heap-snapshot <*worker*> [*filename*]	Save heap snapshot data for worker process <*id*> and save results in file *`name`*`.heapsnapshot`. Saves profiling data to a file you can view with Chrome Dev Tools. See Heap memory profiling for more information.	<*worker*>, a worker specification (see above). *filename*, optional base file name; default is `node.<`*`id`*`>.heapsnapshot`.
objects-start <*worker*>	Start tracking objects on worker process <*id*>.	<*worker*>, a worker specification (see above).
objects-stop <*worker*>	Stop tracking objects on worker process <*id*>.	<*worker*>, a worker specification (see above).
patch <*worker*> <*file*>	Apply patch <*file*> to <*id*> to get custom metrics.	<*worker*>, a worker specification (see above). <*file*>, file name.

slc debug

Run the Node Inspector debugger.

SYNOPSIS

```
slc debug [node-inspector-options] [options] script [script-arguments]
```

The *script* argument is resolved relative to the current working directory. If no such file exists, then the command searches env.PATH.

By default, the command loads the module in the current working directory in the REPL session as `m`. This enables you to call and debug arbitrary functions exported by the current module.

The default mode is to break on the first line of the script. To run immediately on start, use `--no-debug-brk` or click the **Resume** button.

OPTIONS

--cli, -c
CLI mode, do not open browser.

`--debug-brk, -b`
Break on the first line (as with `node --debug-brk`). Default is true.

`--debug-port, -d`
Node/V8 debugger port. Default is 5858.

--web-port, -p, --port
Node Inspector port. Default is 8080.

NODE INSPECTOR OPTIONS

The `slc debug` command passes the following options to the `node-inspector` command.

`--hidden`
Array of files to hide from the UI (breakpoints in these files will be ignored).   Default is `[]`.

`--no-preload`
Do not preload JavaScript files. Use this option for faster startup.

`--save-live-edit`
Save live edit changes to disk (update the edited files). Default is false.

`--stack-trace-limit <n>`
Show <*n*> stack frames on a breakpoint. Default is 50.

`--web-host <port>`

HTTP host on which to listen. Default is 127.0.0.1.

STANDARD OPTIONS

`-h, --help`
Display help information.

`-v, --version`
Display version number.

slc deploy

Deploy a Node application to StrongLoop Process Manager

See also: Deploying applications with slc

SYNOPSIS

```
slc deploy [options] [URL [ package | branch ] ]
```

OPTIONS

`-s, --service` `[service]`
Deploy to service <*service*>, where <*service*> is the service name or ID.   Defaults to the name of the application specified in the `package.json` file. If Process Manager does not have a service with that name, it creates one and deploys the application to it.

`-z, --size` <*n*>
Set size of cluster to <*n*> worker processes before deployment, where <*n*> is a positive integer or the string "cpus," meaning to run one worker per CPU.

When you deploy an application to Process Manager, you give the deployed application instance a name, referred to as the *service name* and indicated in command arguments as `<service>`. By default, it is the `name` property from the application's `package.json`.

Process Manager also automatically generates an integer ID for each application it's managing. Typically, the IDs start with one (1) and are incremented with each application deployed; however, the value of ID is not guaranteed. Always determine it with `slc ctl status` once you've deployed an app.

A service becomes available over the network at `http://hostname:port` where:

- `hostname` is the name of the host running Process Manager
- `port` is 3000 + service ID.

For example, if Process Manager is running on my.host.com, then service ID 1 is available at `http://my.host.com:3001`, service ID 2 at `http://my.host.com:3002`, and so on.

STANDARD OPTIONS

`-h, --help`
Display help information.

`-v, --version`
Display version number.

ARGUMENTS

URL
The URL of the StrongLoop process manager; for example, `http://prod.foo.com:7777/`. Defaults to `http://localhost:8701/`. Both host and port are optional. Host defaults to `localhost` and port defaults to `8701`. If the server requires authentication, the credentials must be part of the URL; see Securing Process Manager.

To connect using SSH and tunnel the HTTP requests over that connection, use the `http+ssh` protocol in *URL*. With `http+ssh`:

- The SSH username defaults to your current user. Override the default by setting the SSH_USER environment variable.
- Authentication defaults to your current SSH agent. Override the default by setting the SSH_KEY environment variable to the path of the existing private key to use.
- SSH port defaults to 22. Override the default by setting the SSH_PORT environment variable.

branch
Deploy the specified branch. The default is a branch named `deploy`.

package
Name of a tar (.tgz) file or npm package to deploy. The default is `../<package_name>-<package_version>.tgz`, where the package name and version come from the `package.json` in the current working directory. **NOTE**: A tar file (.tgz file) must be of the format created by `npm pack`, which is the format created by `slc build` and the Arc Build & Deploy module.

NOTE: If you specify *branch* or *package*, you must also specify *URL*.

EXAMPLES

Deploy the default npm package or git branch to localhost:

```
$ slc deploy
```

Deploy the default npm package or git branch to a remote host:

```
$ slc deploy http://prod1.example.com
```

Deploy to a remote host, on a non-standard port, using authentication:

```
$ slc deploy http://user:pass@prod1.example.com:8765
```

Deploy "production" branch to localhost:

```
$ slc deploy http://localhost production
```

slc env

Error rendering macro 'markdown-url' : Error 503 retrieving server data from URL.

slc lb

This command is deprecated for LoopBack 2.0. Use slc loopback instead.

The `slc` command assumes that the application root directory is always the current directory, even if you change `appRootDir` in code.

Creates LoopBack examples, apps, and workspaces.

Error rendering macro 'markdown-url' : Error 503 retrieving server data from URL.

slc loopback

Create and scaffold a new LoopBack application or add to an existing application.

See also: **See also**: Using LoopBack tools

SYNOPSIS

```
slc loopback[:generator] [options]
```

With no *generator*, prompts for:

- Directory in which to create the application. Press **Enter** to create the application in the current directory.
- Name of the application, defaults to the directory name previously entered.

The tool then creates the standard LoopBack application structure (see Project layout reference for details).

OPTIONS

`--skip-install`
Do not install npm dependencies. Default is false.

`--generators`
List Loopback generators available.

STANDARD OPTIONS

`-h, --help`
Display help information.

`-v, --version`
Display version number.

GENERATORS

```
slc loopback:generator
```

For an existing application, use in the application root directory and provide *generator*, a LoopBack generator:

- ACL generator
- API definition generator
- Application generator
- Boot script generator
- Data source generator
- Middleware generator
- Model generator
- Property generator
- Relation generator
- Remote method generator
- Swagger generator

slc pm

Prerequisite: This command requires Git version 1.8 or higher.

Run StrongLoop Process Manager (PM) to manage your Node applications.

See also: Using Process Manager

After packaging an application with `slc build`, use `slc deploy` to deploy your application to the Process Manager. See Building and deploying and Using Process Manager for more information.

To control Process Manager via HTTP, start it with the `--control` option and specify the desired port. Then you can deploy apps to the PM using `slc deploy` , specifying the host name and port; and you can control PM using slc ctl. To secure access to PM, enable HTTP basic authentication; see Securing Process Manager for more information.

SYNOPSIS

```
slc pm [options]
```

OPTIONS

`-b, --base <basedir>`
Save applications to `basedir`, including both Git repositories and npm packages, as well as the working directories, and any other files required. Default is `.strong-pm` in the current user's home directory (`~/.strong-pm`).

`-C, --control <ctl>`
Listen for control messages on `<ctl>.` Default is ctl.

`-d, --driver <driver>`
Specify application execution driver, either `direct` (the default) or `docker`.

`-l, --listen <port>`
Listen on <*port*> for application deployment and control commands.

`--no-control`
Do not listen for control messages.

`-P, --base-port <port>`
Run applications on port number <*port*> + service ID. Default base port is 3000.

STANDARD OPTIONS

`-h, --help`
Display help information.

`-v, --version`
Display version number.

EXAMPLE

Run StrongLoop Process Manager, listening on port 777:

```
$ slc pm -l 7777
```

`Clone and push an application for a non-production deploy:`

```
$ git clone git@github.com:strongloop/loopback-example-app.git
$ cd loopback-example-app
$ slc deploy http://localhost:7777/repo
```

This brief example installed all the dependencies on the server. In practice, build in app dependencies. Do not install them dynamically at run-time; for example:

```
$ slc build --install --commit
$ slc deploy http://localhost:7777/repo
```

slc pmctl

This command has been renamed to slc ctl. Please update your installation. See Updating to the latest version.

slc pm-install

Use the the `slc pm-install` or `sl-pm-install` command to install StrongLoop Process Manager as a service on Linux distributions that support Upstart or systemd.

By default, the command uses Upstart 1.4 or newer. The default will work, for example on Ubuntu 12.04, 12.10, 13.04, 13.10, 14.04, or 14.10. Otherwise:

- On systems with Upstart 0.6 - 1.3.x (for example Ubuntu 10.04, CentOS, or AWS Linux), use the `--upstart 0.6` option.
- On systems that support systemd instead (for example RHEL 7, Ubuntu 15.04 /15.10, Fedora, or OpenSUSE), use the `--syst`

emd option.

Install StrongLoop Process Manager as an operating system service.

On a system where you have installed StrongLoop with `npm install -g strongloop`, use the `slc pm-install` command. One a production host where you've installed StrongLoop Process Manager with `npm install g strong-pm`, use `sl-pm-install`. The two commands are equivalent.

SYNOPSIS

```
slc pm-install [options]
```

Install StrongLoop Process Manager as a service using the specified options. By default, the command installs the service as an Upstart job using a template for Upstart 1.4 or higher. Use the `--systemd` option to install as a `systemd` service instead. Use the `--upstart <version>` option to specify a different version of Upstart.

See Setting up a production host for more information.

OPTIONS

`-b, --base <base>`
Base directory in which to work. Default is home directory of user running Process Manager; see `--user` option.

`-d, --driver <driver>`
Specify application execution driver, either `direct` (the default) or `docker`.

`-e, --set-env env=val...`
Initial application environment variables, where *env* is name of the enviroment variable, `val` is value. To set multiple variables, enclose them all in quotes as a single argument, and separate each pair with a space, for example "K1=V1 K2=V2 K3=V3".

`-f, --force`
Overwrite existing job file if present.

`--http-auth <creds>`
Enable HTTP basic authentication, requiring the specified credentials for every request sent to the REST API.
<*creds*> has the form <*username*>:<*password*>.

`-j, --job-file <path>`
Path of Upstart job to create (default `/etc/init/strong-pm.conf`)

`-m, --metrics <stats>`
Specify `--metrics` option for supervisor running deployed applications. See Integrating with third-party consoles for more information.

`-n, --dry-run`
Don't write any files.

`-p, --port <port>`
Listen on <*port*>. Default is 8701.

`-P, --base-port <port>`
Run applications on port number <port> + service ID. Default base port is 3000.

`[–systemd | --upstart <version> ]`
Install Process Manager as either:

- A systemd service.
- An Upstart job (Upstart 1.4 is the default). Use this option to specify Upstart 0.6.

`-u, --user`
Run StrongLoop Process Manager as this user. Default is strong-pm.

STANDARD OPTIONS

`-h, --help`
Display help information.

`-v, --version`
Display version number.

slc registry

Switch between different npm registries.

See also: Using multiple package registries

SYNOPSIS

```
slc registry [options] [sub-command] [arguments]
```

When you run this command the first time, it creates a directory in your HOME directory to keep all data files and adds a "default" entry configured to use your current registry as set in `~/.npmrc`.

STANDARD OPTIONS

`-h, --help`
Display help information.

`-v, --version`
Display version number.

SUB-COMMANDS

The `slc registry` command supports the following sub-commands:

- add
- list
- remove
- use
- promote

`add`

```
slc registry add name [url]
```

Create a new registry configuration with the given name. This command runs an interactive wizard that prompts for settings like username and email.

Arguments:

- *name* - Name of the registry to add. Required.
- *url* - URL of the registry server. Optional; if you don't provide as an argument you can specify in response to interactive prompts.

`list`

```
slc registry list
```

List configured registries. This command has no arguments.

```
slc registry remove name
```

Remove the specified registry configuration and related files (for example, cache files).

Argument:

- *name* - Name of the registry to remove. Required. You must have previously added the registry with the `add` command.

`use`

```
slc registry use name
```

Use one of the configured registries previously added with the `add` command.

1. If the registry URL configured in `~/.npmrc` matches one of the registries managed by this tool, the configuration of the matching registry is updated using the values in `~/.npmrc`. This ensures that internal npm settings like authentication tokens are preserved across registry

switches.
2. `~/.npmrc` is modified using the selected registry configuration to let npm use the selected registry.

Arguments:

- *name* - Name of the registry to use. Required.

`promote`

```
slc registry promote --from registry1 --to registry2 package@version
```

Promote a given package version to another registry.

Options:

- `--from registry1` - Name of the registry from which to download the package. You must supply a value for this option if you do not supply a value for the `--to` option. Default is currently active registry.
- `--to registry2` - Name of the registry to which to publish the package. You must supply a value for this option if you do not supply a value for the `--from` option.   Default is currently active registry.

You can omit one of these options; then the command will use the currently active registry.

Argument:

- *package@version* - The name and the version of the package to promote. Example: loopback@1.8.0

The `--version` option displays the version of the registry tool, strong-registry, NOT the version of slc.

ADVANCED USE

The interactive wizard provided by the "add" command covers only a subset of npm options. Change other options using the `npm config` com mand.

1. Switch to the registry you want to configure.

```
$ slc registry use my-registry
```

2. Update the configuration as needed; for example, set "local-address" option.

```
$ npm config set local-address 127.0.0.1
```

3. Next time you switch to a different registry, the changes will be applied to the stored configuration.

```
$ slc registry use default
```

The registry configuration tracks only the npm options that are related to the registry being used. Other options like "browser" or "git" remain always the same, no matter which registry is used.

List of tracked options:

- registry
- username
- email
- proxy
- https-proxy
- local-address
- strict-ssl
- always-auth
- _auth
- _token

For the full list of npm options and their descriptions, see npm-config docs.

slc start

Start application in the background under the Process Manager.

Process Manager will listen for control messages on the default port, 8701.

See also: Running local apps with slc

SYNOPSIS

```
slc start [app]
```

If not specified, the application in the current working directory is started.

STANDARD OPTIONS

`-h, --help`
Display help information.

`-v, --version`
Display version number.

sl-nginx-ctl

Prequisite
You must first install the StrongLoop Nginx Controller. See Configuring Nginx load balancer.

Run StrongLoop NGINX controller.

SYNOPSIS

```
sl-nginx-ctl [options]
```

OPTIONS

`-b, --base <base>`
Base directory in which to work. Default without this option is `.strong-nginx-controller`.

`-c, --control <control>`
Control API endpoint URL. Default without this option is `http://0.0.0.0:0`.

`-l, --listen <endpoint>`
Listen endpoint URL for incoming HTTP traffic. Default without this option is `http://0.0.0.0:8080`.

`-x, --nginx <nginx-path>`
Path to Nginx binary. Default without this option is `/usr/sbin/nginx`.

STANDARD OPTIONS

`-h, --help`
Display help information.

`-v, --version`
Display version number.

sl-nginx-ctl-install

Prequisites
You must first install the StrongLoop Nginx Controller. See Configuring Nginx load balancer.

This command is supported only on Linux distributions that support Upstart or systemd, including RedHat-based distributions (CentOS, RHEL, Fedora, and so on), Debian-based distributions (Ubuntu, Mint, and so on).

Install the StrongLoop Nginx Controller as an operating system service.

SYNOPSIS

```
sl-nginx-ctl-install [options]
```

OPTIONS

`-b, --base <base>`
Base directory in which to work. Default without this option is `.strong-nginx-controller`.

`-c, --control <control>`
Control API endpoint. Default without this option is `http://0.0.0.0:0/`.

`-u, --user <user>`
User account under which to run manager. Default without this option is `strong-nginx-controller`.

`-l, --listen <endpoint>`
Listen endpoint for incoming HTTP traffic. Default without this option is: `http://0.0.0.0:8080`.

`-n, --dry-run`
Don't write any files.

`-j, --job-file <file>`
Path of Upstart job to create. Default without this option is `/etc/init/strong-nginx-controller.conf`.

`-f, --force`
Overwrite existing job file if present.

`-x, --nginx <nginx-path>`
Path to Nginx binary. Default without this option is `/usr/sbin/nginx`.

`--upstart <version>`
Specify the version of Upstart, either 1.4 or 0.6. Default without this option is 1.4.

`--systemd`
Install as a systemd service, not an Upstart job.

OS Service support

The `--systemd` and `--upstart` options are mutually exclusive. If you specify neither, the command installs the service as an Upstart job using a template that assumes Upstart 1.4 or higher.

STANDARD OPTIONS

`-h, --help`
Display help information.

`-v, --version`
Display version number.

sl-pm

This command is available only when you have installed StrongLoop Process Manager with `npm install -g strong-pm` as described in Setting up a production host.

Run StrongLoop Process Manager, when installed as a standalone module (typically on a production host). The options and sub-commands are the same as `slc pm`.

See slc pm for more information.

sl-pmctl

This command is available only when you have installed StrongLoop Process Manager with `npm install -g strong-pm` as described in Setting up a production host.

Control StrongLoop Process Manager. The options and sub-commands are the same as `slc ctl`. See slc ctl for more information.

sl-pm-install

This command is available only when you have installed StrongLoop Process Manager with `npm install -g strong-pm` as described in Setting up a production host.

Install StrongLoop Process Manager as an operating system service. The options and sub-commands are the same as `slc pm-install`.

See slc pm-install for more information.

Arc release notes

The current version of strong-arc is .

The latest version is available on npm.

For a list of the latest commits, see the change log.

Known issues

This is a list of known issues (bugs). Please check this list before reporting a new issue.

Related articles:

High-level issues

General

- StrongLoop Arc does not run in Internet Explorer 8 or Internet Explorer 9. You must use IE 10+.

Composer

Creating models and properties:

- If you choose a base model that is not compatible with the data source, Arc should give error, but does not.
- Model property type field should be populated via loopback-workspace API.
- If you choose a base model that’s a custom model, can’t validate. See issue #34.

Data sources:

- Error messages shown are global, not specific to the data source you are currently editing. That is, you may see error an message stating that you need to install a connector, even when testing connection for another connector that *is* installed.
- Composer should show only data sources that are installed as part of the project.

Build & Deploy

- You cannot deploy applications to Windows systems.
- When using the Oracle connector on a remote host, you must first configure the environment variable (with the `slc ctl env-set` command) and make sure the dynamic libraries from Oracle Instant Client are available for the Node process.

Full list of open issues

This list is from https://github.com/strongloop/strong-arc/issues.

Issue Number	Title	Description

How to report an issue

Related articles:

If you discover a bug or other issue:

1. Review issues that have already been reported:
   - Known issues.
   - GitHub issues at https://github.com/strongloop/strong-arc/issues.
2. If your issue has not been reported, go to GitHub Issues.
3. Click the big green **New** button to report a new issue: https://github.com/strongloop/strong-arc/issues/new.

   - Try to describe the issue in as much detail as possible. If a bug, provide steps to reproduce.
   - Apply the appropriate label: bug, enhancement, question, and so on. If unsure, leave this field empty.
   - Leave the **Assignee** and **Milestone** fields blank.

Thank you!

Upcoming features

Related articles:

The following Arc Composer features are planned for future releases:

- LoopBack API Explorer integrated.
- Scaffold a static web application based on your LoopBack models and data sources.
- Enter data for the models you have created through a form-based web application.
- Canvas view.

The following features will be added in subsequent releases. Specific timing and implementation have not yet been determined.

General

- ACL editor
- Relations editor
- Scopes editor
- Index editor
- Support for validation in LDL

Data sources

- Support for email data sources.
- Manage connector dependencies.

Environment variables

Variable	Description
SSH_KEY	Path to the SSH private key to use when connecting with `http+ssh`. Default is current SSH agent.
SSH_PORT	port to use with `http+ssh`. Default is 22.
SSH_USER	Username when connecting with `http+ssh`. Default is your current user.

STRONG_AGENT_LICENSE	Deprecated. Use STRONGLOOP_LICENSE instead.
STRONGLOOP_CLUSTER	The number of workers the supervisor process will create. Determines the default cluster size if you don't provide an argument to `slc ctl set-size`. See slc ctl set-size for more information.
STRONGLOOP_LICENSE	Your StrongLoop license key(s). Separate multiple license keys with a colon (`:`).
STRONGLOOP_METRICS	Metrics URL. For more information, see Monitoring with slc.
STRONGLOOP_PM	Control channel or Process Manager URL to use. For a remote Process Manager, this must specify the URL on which the Process Manager is listening. See the slc ctl -C option for more information.
STRONGLOOP_PM_HTTP_AUTH	Set to the user name and password for secure access with HTTP authentication with the format STRONGLOOP_PM_HTTP_AUTH=*username*:*password.* See Securing Process Manager for more information.

Other StrongLoop modules

StrongLoop created and maintains many open-source modules. In addition to the LoopBack modules, StrongLoop curates several modules for clustering plus several general-purpose Node modules.

The StrongLoop modules depend on a number of important open-source community modules; see Supporting module reference for a list.

Cluster modules

The following modules enable Node applications to create child processes that all share the same network port. This enables applications to take advantage of multi-core systems:

- Strong-cluster-control - Allows for run-time management of cluster processes.
- Strong-store-cluster - A key-value store accessible to all nodes in a node.js
- Strong-cluster-connect-store - Provides sessions for Connect and Express applications. Manages workers, ensuring the correct number of workers are available; enables you to change the worker pool size without restarting the application.
- Strong-cluster-socket.io-store - Provides an easy solution for running a socket.io server when using node cluster.
- Strong-cluster-tls-store - An implementation of TLS session store using node's native cluster messaging. Additionally, provides several different message queue implementations, including cluster-native messaging.

Other modules

- Node-inspector - Debugging interface for Node.js.
- Strong-mq - An abstraction layer over common message distribution patterns.
- Strong-task-emitter - Performs an arbitrary number of tasks recursively and in parallel and, using an event emitter, passes the results in an asynchronous message.
- Strong-remoting - Makes objects and data in your Node application need to be reachable from other Node processes, browsers, and mobile clients.

Strong Pubsub

Error rendering macro 'markdown-url' : Error 503 retrieving server data from URL.

Zones

This project is deprecated and no longer being actively developed or maintained.

Error rendering macro 'markdown-url' : Error 502 retrieving server data from URL.

Zones examples

This project is deprecated and no longer being actively developed or maintained.

Error rendering macro 'markdown-url' : Error 503 retrieving server data from URL.

Strong Task Emitter

See also the Strong Task Emitter API referenc
e.

Overview

Strong Task Emitter enables you to perform a number of tasks recursively and in parallel. For example, reading all the files in a nested set of directories. It has built-in support for domains by inheriting directly from `EventEmitter`.

Installation

```
$ npm install strong-task-emitter
```

Examples

The following example shows the basic API for a TaskEmitter.

```
var TaskEmitter = require('strong-task-emitter');
var request = require('request');
var results = [];

var te = new TaskEmitter();

te
  .task('request', request, 'http://google.com')
  .task('request', request, 'http://yahoo.com')
  .task('request', request, 'http://apple.com')
  .task('request', request, 'http://youtube.com')
  // listen for when a task completes by providing the task name
  .on('request', function (url, res, body) {
    results.push(Buffer.byteLength(body));
  })
  .on('progress', function (status) {
    console.log(((status.total - status.remaining) / status.total) * 100 + '%',
'complete');
  })
  .on('error', function (err) {
    console.log('error', err);
  })
  .on('done', function () {
    console.log('Total size of all homepages', results.reduce(function (a, b) {
      return a + b;
    }), 'bytes');
  });
```

The next example highlights how to use TaskEmitter to simplify recursive asynchronous operations. The following code recursively walks a social network over HTTP. All requests run in parallel.

```
var TaskEmitter = require('strong-task-emitter');
var request = require('request');
var socialNetwork = [];

var te = new TaskEmitter();

te
  // specify a task name
  .task('friends', fetch, 'me')
  .on('friends', function (user, url) {
    if(url !== 'me') {
      socialNetwork.push(user);
    }

    user.friendIds.forEach(function (id) {
      this.task('friends', fetch, 'users/' + id)
    }.bind(this));
  })
  .on('done', function () {
    console.log('There are a total of %n people in my network', socialNetwork.length);
  });

function fetch(url, fn) {
  request({
    url: 'http://my-api.com/' + url,
    json: true,
    method: 'GET'
  }, fn);
}
```

Extending TaskEmitter

`TaskEmitter` is designed to be a base class which you can easily extend with sub-classes. The following example shows a class that inherits from `TaskEmitter` and provides recursive directory walking and file loading.

```
var TaskEmitter = require('strong-task-emitter');
var fs = require('fs');
var path = require('path');
var inherits = require('util').inherits;

function Loader() {
  TaskEmitter.call(this);

  this.path = path;
  this.files = {};

  this.on('readdir', function (p, files) {
    files.forEach(function (f) {
      this.task(fs, 'stat', path);
    }.bind(this));
  });

  this.on('stat', function (file, stat) {
    if(stat.isDirectory()) {
      this.task(fs, 'readdir', file);
    } else {
      this.task(fs, 'readFile', file, path.extname(file) === '.txt' ? 'utf-8' : null);
    }
  });

  this.on('readFile', function (path, encoding, data) {
    this.files[path] = data;
  });
}

inherits(Loader, TaskEmitter);

Loader.prototype.load = function (path, fn) {
  if(fn) {
    // error events are handled if a task callback ever is called
    // with a first argument that is not falsy
    this.on('error', fn);

    // once all tasks are complete the done event is emitted
    this.on('done', function () {
      fn(null, this.files);
    });
  }

  this.task(fs, 'readdir', path);
}

// usage
var l = new Loader();

l.load('sample-files', function (err, files) {
  console.log(err || files);
});
```

Strong Task Emitter API

Error rendering macro 'markdown-url' : Error 503 retrieving server data from URL.

View docs for strong-task-emitter in GitHub

Strong remoting

See also Strong remoting API

Error rendering macro 'markdown-url' : Error 503 retrieving server data from URL.

...

Supporting module reference

StrongLoop uses a set of community modules that provide crucial functionality, as described in the following table.

Module documentation	NPM Package	Description
Express	Express	Web application framework for building single-page, multi-page, and hybrid web applications.
Connect	Connect	Extensible HTTP server framework using "plugins" known as *middleware*.
Passport	Passport	Authentication middleware for Node.
Mongoose	Mongoose	MongoDB object modeling.
Async	Async	Functions for working with asynchronous JavaScript such as map, reduce, filter, each, and so on, plus common patterns for asynchronous control flow (parallel, series, waterfall).
• Q README • Wiki	Q	Library for promises. If a function cannot return a value or throw an exception without blocking, it can return a promise instead. A promise is an object that represents the return value or the thrown exception that the function may eventually provide. A promise can also be used as a proxy for a remote object to overcome latency.
Request	Request	Request is designed to be the simplest way possible to make http calls. It supports HTTPS and follows redirects by default.
Socket.IO	Socket.IO	Makes WebSockets and realtime apps possible in browsers and mobile devices. Enhances WebSockets by providing built-in multiplexing, horizontal scalability, automatic JSON encoding/decoding, and more.
Engine.IO	Engine.IO	Implementation of transport-based cross-browser/cross-device bi-directional communication layer for Socket.IO.
Node Inspector	Node-inspector	Debugger interface for Node using the Blink Developer Tools (formerly WebKit Web Inspector).
Heap-dump	Heapdump	Provides V8 heap dumps to help track down memory leaks and memory consumption issues.