Documentation

StrongLoop Suite documentation

StrongLoop Suite includes:

- LoopBack, an open-source mobile backend framework for Node.js.
- StrongNode, professional support for Node.js, plus cluster management and other powerful modules.
- StrongOps, a built-in monitoring and management console.

To get up and running quickly, see Getting started.

What's new

LoopBack now enables you to send push notifications to mobile apps. See Creating push notifications for details. Client SDKs have been updated to support push notifications:

- Android SDK (version 1.2)
- iOS SDK (version 1.2)

StrongLoop now supports the Digital Ocean cloud platform.

See What's new for a complete list.

Command-line reference (slc)

- Command syntax
- Getting the latest version of slc

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

Command syntax

```
slc [options] [command] [args]
slc <npm-command> [npm-command-args]
slc <script-file> [args]
```

The second and third command forms are short forms of the `npm` and `run` commands described below. An unrecognized `slc` command is passed to `npm`. Any other unrecognized command is passed to the run command.

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

Command	Description
clusterctl	Control a clustered application.
create	Create Node script boilerplate.
debug	Debug module with Node Inspector.
env	Display information about the Node runtime environment.
example	Create example application.

help	Display help information.
lb	Create LoopBack examples, apps, models, and workspaces.
npm	Run an npm command.
run	Run an application script.
strongops	Register with StrongOps.
version	Display version information.

All commands provide the following option:

- `-h`, `--help`: print usage information for the specified command, or for slc when used without a command.

Getting the latest version of slc

slc clusterctl

slc clusterctl – Control a clustered application

Control an application that has been clustered with strong-cluster-control.

SYNOPSIS

```
slc clusterctl [options] [command]
```

OPTIONS

`-h`, **`--help`**:
print usage information

`-p`, **`--path`**, **`--port <path>`**:
cluster control socket to connect to, defaults to **`./clusterctl`**

`command`:
one of the supported commands, see below (defaults to **`status`**)

Supported commands are:

`status`: report status of cluster workers, the default command

`set-size`: set-size N, set cluster size to N workers

`stop`: stop the cluster controller

`restart`: restart all workers

Useful mostly for debugging:

`disconnect`: disconnect all workers

`fork`: fork one worker

Examples

Run an app in the local directory, with two workers, and see status:

```
slc example urlsaver my-url
    cd my-url
    slc run . --size=cpus &
    slc clusterctl # => print status of workers
```

Get status on an app at a specific path:

```
slc clusterctl -p /apps/my-app/clusterctl status
```

slc create

slc create – Create node script boilerplate

SYNOPSIS

```
slc create <type> <name> [...]
```

OPTIONS

`-h`, **`--help`**:
print usage information

`--no-install`:
skip the **`slc npm install`** step after boilerplate is created

`name`:
name of boiler-plate to create

`...`:
options specific to type, see below

Types

`web`:
simple express app with optional mongoose support

`package`:
npm module with package.json and typical layout

`module`:
node single-file class module in ./lib

`cli`:
empty cli program (no options)

Web options

`-E`, **`--disable-express`**:
disable express support

`-c`, **`--cors`**:
enable CORS for routes

`-m`, **`--mongoose`**:
add mongoose/mongodb support

`-r`, **`--rest`**:
add REST apis for mongoose models

`-t`, **`--title <title>`**:
set the title of your application

Package options

`-i`, **`--inherit-from`**:
module to inherit from, default: EventEmitter

Module options

`-i`, **`--inherit-from`**:
module to inherit from, default: EventEmitter

`-t`, **`--test`**:
included a generated test

`-s`, **`--stream <type>`**:
implement a stream interface (readable, writeable, duplex, transform)

slc debug

slc debug – Debug module with node-inspector

SYNOPSIS

```
slc debug [options] [module]
```

OPTIONS

`-h`, **`--help`**:
print usage information

`-c`, **`--cli`**:
prevent auto opening web browser

`-p`, **`--port <port>`**:
start debugger on port

`-s`, **`--suspend`**:
suspend on first line

`-d`, **`--debug-port <port>`**:
use an alternate debug port

`module`:
module to debug.

Use `.` to debug the application in the current directory.

If no module is specified, then slc will start REPL and load the module in the current directory.

slc env

slc env – Print runtime environment

Prints information about node run-time from the process module (paths, platform, config, execPath, and features). The output can be limited by using one or more selectors, see examples.

SYNOPSIS

```
slc env [options] [selectors...]
```

OPTIONS

`-h`, **`--help`**:
print usage information

EXAMPLES

Print zlib version:

```
% slc env versions zlib
    { versions: { zlib: '1.2.3' } }
```

Print paths:

```
% slc env paths
    { paths:
       { link:
          { node: '/usr/local/bin/node',
            npm: '/usr/local/bin/npm',
            slc: '/usr/local/bin/slc' },
         exec:
          { node: '/usr/local/stow/installed-node/bin/node',
            npm: '/usr/local/lib/node_modules/slc/node_modules/npm/bin/npm-cli.js',
            slc: '/usr/local/lib/node_modules/slc/bin/slc',
            slcNpm: '/usr/local/lib/node_modules/slc/node_modules/.bin/npm',
            nodeNpm: '/usr/local/stow/installed-node/bin/npm' } } }
```

slc example

slc example – Create example applications

Create StrongLoop example applications.

SYNOPSIS

```
slc example [type [path]] [flags]
```

OPTIONS

`-h`, **`--help`**:
print usage information

`--no-install`:
skip the **`slc npm install`** step after example is created

`type`:
one of the supported example types, see below (defaults to **`suite`**)

`path`:
where to create the example (optional, examples have default names)

Supported example types are:

`suite`: LoopBack API server, demonstrates LoopBack integrated with StrongNode, this is the default type

`chat`: web chat application, demonstrates socket.io and express integrated with StrongNode clustering

`urlsaver`: document storage API server, demonstrates creating an API server with express, request, Q, and StrongNode clustering

`blog`: weblog application, demonstates an express web app integrated with StrongNode clustering

Examples

Create the suite example:

```
slc example
```

Create the chat example, in a specific path:

```
slc example chat example/my-chat
```

slc help

slc help – Print usage information

Print usage for slc, or for a specific slc command.

SYNOPSIS

```
slc help [options] [command]
```

OPTIONS

`-h`, **`--help`**:
print usage information

`command`:
command to print usage for

slc lb

Creates LoopBack examples, apps, and workspaces.

slc lb – Create LoopBack examples, apps, and workspaces

SYNOPSIS

```
slc lb <command> <arg> [flags]
```

COMMANDS

Supported `lb` commands are:

`workspace [name]`:
Initialize a workspace as a new empty directory with an optional name. The default name is “loopback-workspace”.

```
$ slc lb workspace my-loopback-workspace
```

`project <name> [--no-install]`:
Create a LoopBack application in a new directory within a workspace using the given name. The argument is required. With **`--no-install`**, will not install the npm dependencies.

```
$ cd my-loopback-workspace
$ slc lb project my-app
$ slc run my-app # to run the app
```

`model <name>`:
Create a model in an existing LoopBack application using the given name. If you provide the **`-i`** or **`--interactive`** flags, you will be prompted through a model configuration. The argument is required.

```
$ cd my-app
$ slc lb model product -i
```

`datasource <name>`:
Create a datasource in an existing LoopBack application using the given name.
You must supply a connector name using the **`--connector`** option.

```
$ cd my-app
$ slc lb datasource mongo --connector mongodb
```

`acl`:
Add a new permission to an existing model or to all models. Creates an ACL entry
Use options described below to configure the exact permission.

ACL Options

`--model <name>`:
Specify the model name to apply the new permissions against.
You must supply either the **`--model`** or **`--all-models`** argument.

`--all-models`:
Apply the permission to all models. You must supply either the **`--model`** or **`--all-models`** argument.

ACL Access Types

`--all`:
Set the access type to the wildcard. This matches **`read`**, **`write`** and **`execute`**.

`--read`:
Set the access type to **`READ`**.

`--write`:
Set the access type to **`WRITE`**.

`--execute`:
Set the access type to **`EXECUTE`**.

ACL Properties and Methods

`--property`:
Specify a specific property to apply the permission to. Defaults to all.

`--method`:
Specifiy a specific method to apply the permission to. Defaults to all.

ACL Role Identifiers

`--owner`:
Only apply this permission to users who own the specified model instance.

`--related`:
Any user with a relationship to the object

`--authenticated`:
Authenticated users

`--unauthenticated`:
Unauthenticated users

`--everyone`:
All users

ACL Permissions

`--alarm`:
Generate an alarm, in a system dependent way, the access

`--allow`:

Explicitly grants access to the model.

`--deny`:
Explicitly denies access to the model.

`--audit`:
Log, in a system dependent way, the access specified in the permissions component of the ACL entry.

ACL Notes

you may only supply a single access type

you may only supply a single role identifier

you may only supply a single permission

ACL Examples

```
# disable all access to all models
# note: other permissions will override this
$ slc lb acl --deny --all-models --everyone --all

# allow authors to edit posts
$ slc lb acl --model post --allow --owner --all

# only allow owners to access objects
$ slc acl --deny --all-models --everyone --all
$ slc acl --allow --all-models --owner --all
```

OPTIONS

`-h`, **`--help`**:
Display this help text.

`-i`, **`--interactive`**:
Run an interactive model creation. Only supported with the model command.

`--data-source <name>`:
Supply a custom data source when creating a model. Defaults to “db”.

`--private`:
Do not expose the model’s API remotely.

`-c`, **`--connector`**:
Specify the connector name when creating a datasource (required).

slc npm

slc npm – Run an npm command

SYNOPSIS

```
slc npm [options] <npm-command> [npm-command-args]
```

OPTIONS

`-h`, **`--help`**:
print usage information

Supports all npm commands, the most commonly used are:

`install`:
install a package from the StrongLoop registry, or from npmjs.org if a

supported version of the package does not exist

`ls`:
list packages with their support status

`update`:
update packages to latest

slc run

slc run – Run a node script

SYNOPSIS

```
slc run <script-file> [script-options]
slc run
```

OPTIONS

`-h`, **`--help`**:
print usage information

`script-file`:
node script file to run

Use `.` to run the application in the current directory.

If no module is specified, then slc will start REPL and load the module in the current directory.

For more information on Node REPL, see REPL.

slc strongops

slc strongops – Register with StrongOps

The "strongops" command registers users with the StrongOps platform. The simplest usage is `slc strongops`. The command prompts for user name, email and a password and then saves the credentials to ./strongloop.json. If no command line options are given for name or email, the strongops command tries to find defaults from `~/.gitconfig` (name and email) and then `~/.npmrc` (email). However, when you specify valid options on the command line, there are no interactive prompts for the data specified on the command line.

SYNOPSIS

```
slc strongops [options]
```

OPTIONS

`--name`:
Specify the full user name, e.g.: **`--name 'Bob Roberts'`** (quotes are needed as the full name contains a space).

`--email`:
Specify the email address, e.g.: **`--email someone@strongloop.com`**.

`--password`:
Specify your StrongOps password, e.g.: **`--password 12345678`**

`--nosave`:
Prevent saving of StrongOps account credentials, this overrides any save option.

`--local`:
Saves StrongOps account credentials in a **`./strongloop.json`** file. This is set by default, if no other save options exist.

`--package`:
Saves StrongOps account credentials in **`./package.json`** file, if the file exists.

`--global`:
Saves StrongOps account credentials in a **`~/strongloop.json`** file.

`--register`:
Skips attempt to login and registers a new user. By default, the command logs users into StrongOps.

`--saveall`:
Saves StrongOps account credentials to **`./strongloop.json`**, **`./package.json`** and **`~/strongloop.json`** files.

`--nocolors`:
Does not use ANSI colors in the responses - this is the default.

`--colors`: Use ANSI colors. If used with **`--nocolors`**, overrides **`--nocolors`**.

See strong-agent for a description of the format of the `package.json` and `strongloop.json` properties used by StrongOps.

EXAMPLES

The following will have strongops prompt for the name, email and password. The JSON containing the strongops credentials are sent to standard output:

```
$ slc strongops
```

Next, the strongops credentials are saved to the local `./package.json` file:

```
$ slc strongops --package
```

By using the options for name, email and password, you can have a non-interactive registration, e.g. that will not prompt:

```
$ slc strongops --name "Bill Williams" --email "bw@strongops.com" --password "12345678"
```

Using the saveall option causes strongops to save the credentials to `./package.json`, `./strongloop.json` and `~/strongloop.json`:

```
$ slc strongops --saveall
```

Applications can be set up so that whether Strong Ops profiling occurs is configured at deploy/run time, based on whether the credentials have been saved. To do this, call `profile()` with no arguments, as below:

```
// app.js
    // Call profile() before any modules which need profiling are
    // required. In the absence of credentials, this is effectively
    // a null op.
    require('strong-agent').profile();

    // ... rest of application is required ...
```

slc version

slc version – Print versions

Print slc and Node.js versions.

SYNOPSIS

```
slc version [options]
```

OPTIONS

-h, **--help**:
print usage information

sl-run

SYNOPSIS

Run an application with StrongOps profiling and supervision. See Using Strong Supervisor for more information.

```
sl-run [options] [app [app-options...]]
```

You can also use `slr` as an alias of `sl-run`.

Optional argument *app* is a node file to run or a package directory. The default value is ".", the current working directory. Packages are run by requiring the first of the following in the current directory:

- `server.js`
- `app.js`
- `main` property specified in the `package.json` file.
- `index.js`

OPTIONS

-h, --help
Print usage information.

-v, --version
Print runner version and exit.

-d, --detach
Detach master from terminal to run as a daemon (default is to not detach).

-l, --log *file*
When detaching, redirect terminal output to *file*, in the app's working directory if *file* path is not absolute (default is supervisor.log)

-p, --pid *file*
Write supervisor's process ID to *file*, failing if *file* already has a valid process ID in it (default is not to)

--size n
Set the cluster size (default is "CPUs").

Cluster size is one of:

- An integer specifying the number of workers to run
- A string containing "cpu" to run one worker per CPU.
- The string "off" to run unclustered, in which case the app will not be supervisable or controllable, but will be monitored.

--no-profile
Do not profile with StrongOps (default is to profile if registration data is found).