out

Contents

Packages

```
package MyPac {
    fn quit {
        parent::quit();
    }
}
```

Translates into:

```
package MyPac{function quit(){parent::quit();}};
```

Prefix the package with `active` to automatically append `activatePackage(MyPac);`

Classes

```
class Foo {
    author = "RoboCop";

    constructor(n) {
        this.n = n;
    }

    fn inc {
        this.n++;
    }
}

x = Foo::create();
x.inc();
```

Translates into:

```
new ScriptObject(Foo){author=" RoboCop";};
function Foo::create(%n){
    %this = new ScriptObject(){class="Foo";};
    %this.n = %n;
    return %this;
}
function Foo::inc(%this){%this.n++;}
%x = Foo::create();
%x.inc();
```

Image states

```
data ShapeBaseImageData MyImage {
    state Activate {
        sound = EquipWeaponSound;
        timeoutValue = 0.25;
        transitionOnTimeout = "Ready";
    }
```

```
    state Ready {
        ...
    }
    ...
}

...
```

Overloaded "array" access

Instead of being equivalent to `xy`, `x[y]` translates to `x.__getItem(y)`. See below if array access is needed.

Fenced TorqueScript code

```
for i in 0..mg.numMembers {
    `%mb = %mg.member[%i];`
    mb.delete();
}
```

Sugar for collection types

```
a = ["x", 6, foo];
echo(a[1]);
for v in a.iter() { echo(v); }
```

Translates into:

```
%a = new ScriptObject() {
    class = "Vec"; // should be namespaced more
    length = 3;
    value0 = "x";
    value1 = 6;
    value2 = %foo;
};
echo(%a.__getItem(1));
// ...
```

Lever

Lever is a tool designed to make Blockland development better. It consists of three things:

- The original Lever - a transpiler that generates TorqueScript from a nicer language,
- liblever, a TorqueScript support library enabling the additional features of the Lever syntax, and
- A CLI for leveraging the Lever transpiler, which also integrates some tools to streamline development.

Getting Set Up

1. Install NodeJS.
2. Get the code - Clone this repository or use the Download ZIP button above.
3. Install the package: Extract the ZIP if necessary, then call `npm install -g` from the lever directory. You will probably need administrative privileges to do this (Right click cmd -> Launch as Administrator, or `sudo`)

Basic Usage

`lever [command] [args]`

Global switches:

- -s, –silent: Suppress all non-erroneous console output.
- -h, –help: Get help with the given command. (unimplemented)

Commands:

- `compile`: Compile a given set of files. If no valid command is supplied, defaults to this.
- `-c, --compact`: Omit all unnecessary whitespace.
- `-o, --out`: Write all output to the given file instead of individual files.
- `init | new`: Create a skeleton for a new add-on. (unimplemented)
- `-t, --title`: Pre-supply a title for the add-on. Defaults to the current working directory's name.
- `-a, --author`: Pre-supply an author for the add-on. Defaults to the USERNAME environment variable.
- `-d, --desc`: Pre-supply a description for the add-on.
- `build | package`: Package the current project into an Add-On. (unimplemented)
- `-c, --compact`: As with compile.
- `-o, --out`: Output to the given file instead of the default zip.

Syntax

Basic Syntax

Local variables do not have `%` prepended to them - they are plain. `client` is the same as `%client` in raw TorqueScript. Lever also uses the same standard operands as TorqueScript - including the Torque-specific string catenation operands `@` `SPC` `TAB` and `NL`.

Functions

The following are equivalent:

```
fn myFunc {
    echo("Hello world!");
}

fn myFunc() {
    echo("Hello world!");
}
```

Arguments are supplied much as in normal TorqueScript:

```
fn myFunc(a, b) {
    echo(a @ " " @ b @ "!");
}
```

Lever also supports anonymous functions:

```
schedule(1000, 0, fn(a) { echo(a); }, "Hello world!");
```

Packages

```
package MyPackage {

};
```

Prepending `active` will activate the package by default:

```
active package MyPackage {

};
```

Parenting is done by simply calling `Parent` regardless of the function:

```
active package MyPackage {
    fn myFunc(a, b) {
        Parent("Hello", "world");
    }
};
```

Classes

DISCLAIMER: Example here is slightly broken - class properties cannot be declared like this yet.

Lever has closer to first-class support for Classes:

```
class Foo {
    author = "RoboCop";

    fn onNew(n) {
        this.n = n;
    }

    fn inc {
        this.n++;
    }
}

x = Foo::create();
x.inc();
```

Fenced TorqueScript Code

If you need to just write some raw TorqueScript, you can do so:

```
for i in 0..mg.numMembers {
    `%mb = %mg.member[%i]`;
    messageClient(mb, '', "nope");
}
```

API

Lever supports a robust API through NodeJS:

`require("lever")(source, options)`

Current options:

- compact: Remove unnecessary whitespace. Defaults to false (pass switch `-c` or `--compact`).

//TODO: Improve API.