# Getting Started with iRules LX, Part 1: Introduction & Conceptual Overview

**Eric Flores, 2016-06-06**

## What is iRules LX?

When BIG-IP TMOS 12.1.0 released a couple weeks ago, Danny wrote an intro piece to a new feature called iRules LX. iRules LX is the next generation of iRules that allows you to programmatically extend the functionality of BIG-IP with Node.js. In this article series, you will learn the basics of what iRules LX is, some use cases, how the components fit together, troubleshooting techniques, and some introductory code solutions.

## What is Node.js ?

Node.js provides an environment for running JavaScript independent of a web browser. With Node.js you can write both clients and servers, but it is mostly used for writing servers. There are 2 main componets to Node.js -

#### Google v8 JavaScript Engine

Node.js starts with the same open source JavaScript engine used in the Google Chrome web browser. v8 has a JIT complier that optimizes JavaScript code to allow it to run incrediable fast. It also has an optimizing complier that, during run time, further optimizes the code to increase the performance up to 200 times faster.

#### LibUV Asynchronous I/O Library

A JavaScript engine by itself would not allow a full set of I/O operations that are needed for a server (you dont want your web browser accessing the file system of computer after all). LubUV gives Node.js the capability to have those I/O operations. Because v8 is a single threaded process, I/O (which have the most latency) would normally be blocking; that is, the CPU would be doing nothing while waiting from a response of the I/O operation. LibUV allows for the v8 to continue running JavaScript while waiting for the response, maximizing the use of CPU cycles and providing great concurrency.

## Why Node.js?

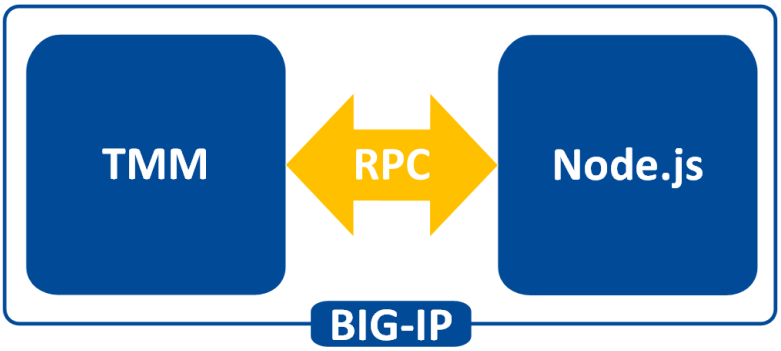
Some might ask "Why was Node.js chosen?" While there is more than one reason, the biggest factor is that you can use JavaScript. JavaScript has been a staple of web development for almost 2 decades and is a common skillset amongst developers. Also, Node.js has a vibrant community that not only contributes to the Node.js core itself, but also a plethora of people writing libraries (called modules). As of May 2016 there are over 280,000 Node.js modules available in the Node Package Manager (NPM) registry, so chances are you can find a library for almost any use case. An NPM client is included with distributions of Node.js.

For more information about the specifics of Node.js on BIG-IP, please refer to AskF5 Solution Article SOL16221101.

## iRules LX Conceptual Operation

Unlike iRules which has a TCL interpreter running inside the Traffic Management Micro-kernel (TMM), Node.js is running as a stand alone processes outside of TMM. Having Node.js outside of TMM is an advantage in this case because it allow F5 to keep the Node.js version more current than if it was integrated into TMM.

In order to get data from TMM to the Node.js processes, a Remote Procedural Call (RPC) is used so that TMM can send data to Node.js and tell it what "procedure" to run. The conceptual model of iRules LX looks something like this -




To make this call to Node.js , we still need iRules TCL. We will cover the coding much more extensively in a few days, so dont get too wrapped up in understanding all the new commands just yet.

Here is an example of TCL code for doing RPC to Node.js -

```
when HTTP_REQUEST {
   set rpc_hdl [ILX::init my_plugin my_extension]
   set result [ILX::call $rpc_hdl my_method $arg1]

   if { $result eq "yes" } {
      # Do something
   } else {
      # Do something else
   }
}
```

In line 2 you can see with the `ILX::init` command that we establish an RPC handle to a specific Node.js plugin and extension (more about those tomorrow). Then in line 3 we use the `ILX::call` command with our handle to call the method `my_method` (our "remote procedure") in our Node.js process and we send it the data in variable `arg1`. The result of the call will be stored in the variable `result`. With that we can then evaluate the result as we do in lines 5-9.

Here we have an example of Node.js code using the iRules LX API we provide -

```
ilx.addMethod('my_method', function(req, res){
  var data = req.params()[0];

  <…ADDITIONAL_PROCESSING…>

  res.reply('<some_reply>');
});
```

You can see the first argument `my_method` in `addMethod` is the name of our method that we called from TCL. The second argument is our callback function that will get executed when we call `my_method`. On line 2 we get the data we sent over from TCL and in lines 3-5 we perform whatever actions we wish with JavaScript. Then with line 6 we send our result back to TCL.

## Use Cases

While TCL programming is still needed for performing many of the tasks, you can offload the most complicated portions to Node.js using any number of the available modules. Some of the things that are great candidates for this are -

**Sideband connections -** Anyone that has used sideband in iRules TCL knows how difficult it is to implement a protocol from scratch, even if it is only part of the protocol. Simply by downloading the appropriate module with NPM, you can easily accomplish many of the following with ease -

- Database lookups - SQL, LDAP, Memcached, Redis
- HTTP and API calls - RESTful APIs, CAPTCHA, SSO protocols such as SAML, etc.

**Data parsing -**

- JSON - Parser built natively into JavaScript
- XML
- Binary protocols

In the next installment we will be covering iRules LX configuration and workflow.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

| | | | |
|---|---|---|---|
| F5 Networks, Inc.<br>Corporate Headquarters<br>info@f5.com | F5 Networks<br>Asia-Pacific<br>apacinfo@f5.com | F5 Networks Ltd.<br>Europe/Middle-East/Africa<br>emeainfo@f5.com | F5 Networks<br>Japan K.K.<br>f5j-info@f5.com |