BIG-IP® iRulesLX User Guide

Version 12.1

Table of Contents

About iRulesLX

What is fundamentally different about iRulesLX? iRulesLX takes advantage of the capabilities of Node.js to enhance the data plane programmability of a BIG-IP® system. To enhance the programmability aspects of iRules®, iRulesLX adds a mechanism to invoke programs in Node.js. Node.js is well-suited to a BIG-IP system, providing a single-threaded environment in which to run programs, taking advantage of asynchronous behavior, and offering a wide range of packages for download. As a developer, the resources of a vast community can help you add functionality to your Node.js applications, while reducing the development effort.

Why Node.js?

What benefits will you realize with Node.js as a development platform in the next generation of iRules®? The most obvious benefit is that Node.js is written in JavaScript, which is a popular and well-known language among web application developers. For an application developer, not having to learn a new language is one less hurdle to clear should you decide to develop for the iRulesLX platform. Node.js also offers a couple of features that make it suitable for the Traffic Management Operating System® (TMOS®) platform.

Node.js runs your code in a single-threaded process. The asynchronous behavior of Node.js improves the runtime performance of server-side JavaScript. If your code depends on the completion of a task like I/O, the runtime places the task in a queue and continues processing your code. Keep in mind that a Node.js process will block in certain circumstances, but Node.js handles I/O requests efficiently. When the I/O task completes, a callback function runs on the results of the task, providing the second noteworthy feature of Node.js.

Node.js runs callback functions upon receiving an event notification, such as the completion of the I/O task mentioned previously that resulted in an event being emitted. In your code, you provide a callback function as a parameter to a function. Because JavaScript supports first-class functions, you can pass a callback as a parameter, and the Node.js runtime will run that function on completion of the I/O task.

By using Node.js, you can enhance iRules functionality and incorporate additional features, such as the use of relational or document databases. While Node.js may not be the best solution in all situations, Node.js offers reasonably high performance capabilities to JavaScript applications, enabling new functionality in iRules.

About Node.js development

iRulesLX adds functionality that enables you to make a call to a Node.js process, run JavaScript code in the Node.js process, and then return the results to the caller. For most extensions that you create, there are two individual yet related tasks in the development process. The first task is the call that invokes the Node.js extension.

The TCL sample code specifies a `when` command and a block of code to run when a specific event occurs. You can call any supported iRules® event, and this sample shows an `HTTP_REQUEST` event on a BIG-IP® system. Within the curly braces, the first line of code uses the iRules® command `set` to create a variable, which is the handle for the call. In the handle variable, specify the endpoint and the name of the extension that runs in the Node.js process. The second line of the sample uses the `set` command to create a variable that holds the output of the call.

```
# Sample code for any iRules event

when HTTP_REQUEST {
    # Typical iRule / TCL code
    set hndl [ILX::init "/Common/isc" "Ext1"]
    set result [ILX::call $hndl func arg]
    # Process the result
}
```

The second task to complete for an iRulesLX extension is the Node.js code itself. The first line of the JavaScript sample assigns the `f5-nodejs` package to a variable. If you need other packages in your extension, you should use the `require` method to load them. The second line of the sample uses the `f5` variable to instantiate an instance of an `ILXServer`. The constructor method creates an ILX server object to listen on a port for an event. In this case, `ilx.listen` listens for an event that is generated when a rule invokes `ILX::call`. The callback function takes the request object and tries to locate a function that matches the function named in the argument string. If a match is found, the callback runs the function and returns the results to the caller.

```
/* Load npm or other custom package */

/* Load the f5-nodejs package */
var f5 = require('f5-nodejs');
var ilx = f5.ILXServer();

/* Listen for calls from iRules */

ilx.addMethod('<function name>', function(req, res)
 {
    /* ... typical JavaScript code ... */
    /* Reply with results */
    res.reply(ret); });
ilx.listen();
```

In the event block in the JavaScript code, you can write code to parse the contents of a packet, connect to other services or databases, or cache data. The `res.reply` statement in the JavaScript code returns the results to the `result` variable in the TCL code block.

About packages and modules

Third-party packages and libraries extend the functionality of Node.js. The node package manager (npm) site lists thousands of packages that you can install and use in Node.js extensions. Common packages that you may want to use in iRules® include parsers for JSON and XML, libraries to consume other services and databases, and distributed memory object caching systems like memcached. In addition to frameworks designed to simplify Node.js application development, packages are available for, among others:

- JSON parsers
- XML parsers
- Memcached
- Redis
- MongoDB
- MySQL

Note:

The version of Node.js in the BIG-IP® system offers full Node.js compatibility and supports the same packages as the version of Node.js that you download from the web site.

About tmsh and the iRulesLX environment

The iRulesLX development environment consists of workspaces, extensions, and rules. To simplify the task of creating workspaces and other directories, iRulesLX includes a set of `tmsh` commands to accomplish many of the tasks related to the creation and maintenance of a development environment. If you want to create a simple workspace with a single extension and a single rule, and then publish the rule and attach it to a virtual server, iRulesLX supports that task through a concise set of `tmsh` commands.

iRulesLX follows the Node.js model to take advantage of the common tools that support Node.js. When you edit a workspace extension directory as part of a development environment, the `tmsh` command creates a `package.json` file in the extension directory. The `package.json` file contains the meta data for a Node.js application, and the file also makes the application a valid node package manager (npm) module. The package manager for Node.js (npm) can use `package.json` to install the application on other BIG-IP® systems.

iRulesLX makes use of staging directories for iRules®. Because of this difference, if you edit and then publish an existing rule, you must follow a different procedure for iRulesLX. In this case, you edit the rule in the workspace, not in a production environment.

About the iRulesLX development environment

The development environment for iRulesLX exists in a conventional directory structure (`/var/`*`ilx`*`/workspaces`). Within the directory structure, individual workspaces are identified by a partition, such as `ltm`, as well as a workspace name. For example, a particular workspace may exist in the following path: `/var/`*`ilx`*`/workspaces/`*`partition`*`/`*`workspace name`*. The complete directory structure for iRulesLX includes the following directories:

- /var/*ilx*/workspaces/*partition*/*workspace name*
- /var/*ilx*/workspaces/*partition*/*workspace name*/extensions
- /var/*ilx*/workspaces/*partition*/*workspace name*/rules

You can use `ssh` to open a shell in the workspace, or use `tmsh` commands to publish a rule and its associated extensions and packages. The rule and its associated extensions and packages are referred to collectively as a plug-in. You publish the plug-in by copying it to the `/var/sdm/plugin_store/plugins` directory. When you publish a plug-in by moving it to a different directory location, you do not actually run the plug-in.

Working in the iRulesLX development environment

As an iRules® developer, you must create a development environment before you can edit and publish a rule using iRulesLX. Complete these steps to create a workspace, publish a rule from the workspace, and attach the rule to a virtual server.

1. From a `tmsh` command prompt, run the following command to create a workspace.

   ```
   create ilx workspace w
   ```

2. Using the same `tmsh` command prompt, run the following command to create a rule in the workspace.

   ```
   edit ilx workspace w rule r
   ```

3. To edit an extension in the workspace, run the following `tmsh` command .

   ```
   edit ilx workspace w extension e
   ```

4. To edit a file in the extension directory, run the following `tmsh` command.

   ```
   edit ilx workspace w extension e file f
   ```

5. To create a plug-in from the developer workspace, run the following `tmsh` command.

   ```
   create ilx plugin p from-workspace w
   ```

6. To attach the rule to a virtual server, run the following `tmsh` command.

   ```
   modify virtual v rule w/r
   ```

You have now created a development environment, a plug-in, and attached the rule to a virtual server.

Republishing in the iRulesLX environment

As an iRules® developer, you must edit a rule in the development environment before you can republish a rule using iRulesLX. Complete the following steps to republish a rule from a workspace.

1. Using a `tmsh` command prompt, run the following command to edit an existing rule in the workspace.

   ```
   edit ilx workspace w rule r
   ```

2. To edit a file in the extension directory, run the following `tmsh` command.

   ```
   edit ilx workspace w extension e file f
   ```

   You may use this particular command as often as necessary to edit files in the extension directory.

3. To republish the modified plug-in, run the following command.

   ```
   modify ilx plugin p reload
   ```

You have now modified and republished the plug-in.

About iRulesLX graphical editor

iRulesLX provides tools to edit Node.js extensions and manage plug-ins. Management tasks include creating, importing, and exporting workspaces, as well as enabling and disabling plug-ins, or modifying the properties of a plug-in. Editing features include the ability to open a file by double-clicking a file in the workspace, line numbering, syntax highlighting, and matching of parentheses and braces within a file. Using the graphical editor, you can produce a plug-in from a workspace and enable or disable a plug-in to run on a BIG-IP® system.

Creating a new workspace

Manage an iRulesLX workspace by using the Traffic Management User Interface (TMUI) to access a BIG-IP® system.

1. Log in to the BIG-IP system with your user name and password.
2. On the Main tab, click **iRules**.
3. Click **LX Workspaces** to display the list of existing iRulesLX workspaces.
   You must provision ILX (`System\Resource Provisioning`) in order to view the ILX menu items.
4. Click the **Create** button.
5. When prompted, type a name for the new workspace.

Deleting a workspace

You can reduce clutter by deleting workspaces that you no longer need or use.

1. On the Main tab, click **iRules**.
2. Click **LX Workspaces** to display the existing workspaces.
3. Select a workspace to delete.
4. Click the **Delete** button.
   When you delete a workspace, you delete the contents of the workspace, as well as the workspace.

Exporting a workspace

To save time and work, you can export a workspace to another BIG-IP® system.

1. On the Main tab, click **iRules**.
2. Click **LX Workspaces** to display the existing workspaces.
3. Select a workspace to export.
4. Click the **Export** button.
   You can also use the export functionality to archive a workspace.

Importing a workspace

To leverage existing iRules®, you can import a workspace from another BIG-IP® system.

1. On the Main tab, click **iRules**.
2. Click **LX Workspaces** to display the existing workspaces.
3. Click the **Import** button.
4. Select a workspace to import.

   When you import a workspace, you must choose the source of the workspace, such as the name of an archive file, a URI that identifies an archive, a workspace, or a plug-in.

Adding a rule in the workspace editor

For individual workspaces, you can use the workspace editor to create a new rule or make changes to an existing rule.

1. Click **Add iRule** to add a rule.

   The workspace editor screen appears when you create or import a workspace, or when you open a workspace to make modifications.
2. To delete any rule, extension, or extension file, select the item and click the **Delete** button.

Adding an extension in the workspace editor

For individual workspaces, you can use the workspace editor to create a new extension or make changes to an existing extension.

1. Click the **Add Extension** button to add an extension to a workspace.

   In this context, an extension consists of scripts, files, or Node.js modules.
2. To delete any rule, extension, or extension file, select the item and click the **Delete** button.

Adding a file to an extension in the workspace editor

When you are working with an extension in the workspace, you can use the workspace editor to add a file to an existing extension.

1. Click the **Add Extension File** button to add a file to an extension.

   You must select an extension to enable the button.
2. To delete any rule, extension, or extension file, select the item and click the **Delete** button.

Reverting to a previous version in the workspace editor

As a convenience, you can revert any unsaved changes to a file and restore the previous version.

1. Click the **Revert File** button.

   If you make a number of changes and then decide not to continue, you can restore the previous version of a file that is open in the editing panel. To undo an individual change to a file, use the Ctrl+Z key combination.
2. To save the changes, rather than revert to the previous saved copy of a file, click the **Save File** button.

   When you save the changes, you are saving the changes to the file open in the editing pane.

Viewing plug-in properties

You can click on a plug-in to view its properties using the LX Plugins screen settings.

1. To reload a plug-in from a workspace, click **Reload from Workspace**.

   You can select a workspace other than the workspace used to originally create the plug-in. The list of available workspaces appears in the drop down list. When you choose to reload the workspace, you incorporate workspace changes into the plug-in. Reloading a workspace integrates any changes you made to a workspace that are not yet part of a plug-in. By creating multiple workspaces, you can implement multiple versions of a plug-in.
2. To view the extensions properties, click on one of the extensions listed in **Extensions**.

Viewing extension properties

You can access any of the available property settings for an extension and make a change, such as entering a value or selecting a value from a drop-down list.

1. To specify a concurrency mode for the extension, select a value from the drop-down list.

   The **Dedicated** setting specifies a separate Node.js process for each provisioned Traffic Management Microkernel (TMM). **Single** specifies a single Node.js process for all TMM processes.
2. To specify the maximum number of restarts for an extension, type a value in the **Maximum Restarts** field.

   Specifies the maximum failures for an extension process before the system abandons efforts to restart the process. The default value is 5.
3. To specify a time interval for maximum restarts, type a value in the **Restart Interval** field.

   Specifies the time, in seconds, that the maximum number of restarts (Maximum Restarts) can occur. The default value is 60 seconds.
4. To enable debugging, check the **EnableDebug** setting.

   Enable or disable debug mode for the extension. You must restart the plugin for the setting to take effect.
5. To specify a range of port numbers, type a value for the **Debug Port Range Low** and **Debug Port Range High** fields.

After you enable debugging, the iRulesLX process searches for an available port to attach the node inspector. The low value represents the low end of the port range that iRulesLX will try, and the high value represents the high end of the port range. iRulesLX starts with the low end port number and increments the value until it locates an available port or reaches the high end port.

Creating an iRulesLX plug-in

After you save the workspace files, create a plug-in from the workspace by navigating to the LX Plugins screen and following the steps to create a plug-in.

1. On the Main tab, select **LX Plugins**, and click the **Create** button.
2. For the new plug-in, type a name for the plug-in and select the corresponding workspace for the plug-in from the drop-down list. You can provide a description for the plug-in you are creating, although the description is optional.
3. Click **Finished** to create the plug-in.

   Click **Repeat** to create the plug-in if you want to create a similar plug-in with a different name. The repeat feature uses the same settings.

Once you have created a plug-in, you can begin using it by attaching the corresponding rule to a virtual server.

Legal Notices

Legal notices

Publication Date

This document was published on May 9, 2016.

Publication Number

MAN-0591-01

Copyright

Trademarks

Trademarks

Patents

Export Regulation Notice

RF Interference Warning

FCC Compliance

may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

Any modifications to this device, unless expressly approved by the manufacturer, can void the user's authority to operate this equipment under part 15 of the FCC rules.

Canadian Regulatory Compliance

This Class A digital apparatus complies with Canadian ICES-003.

Standards Compliance

This product conforms to the IEC, European Union, ANSI/UL and Canadian CSA standards applicable to Information Technology products at the time of manufacture.

Index