Product Documentation

LightWave Server Documentation

LightWave Server Release 1.3.0
26 Jan 2026

Table of Contents

Welcome to the documentation for LightWave Server™ by NuWave Technologies. You may navigate through the documentation using any of these methods:

- The links in the sidebar to the left.
- The quick links shown below.
- The search bar in the page header.

If you are a new LightWave user, please start with How To Begin. If you have comments on this documentation or questions about the product that are not answered here, feel free to submit your comment or question at our Support Center.

Getting Started

For Developers

For System Administrators

How To Begin

This section describes how to proceed if you are a first time LightWave Server™ user. LightWave Server users typically fall into one of the categories shown below. The sections below describe how each category of user should proceed.

- Administrators - An administrator is anyone who is responsible for installing and managing the LightWave Server processes. Systems/Operations managers usually fall into this category.
- Developers - A developer is anyone responsible for building applications that use LightWave Server. Developers often need to manage LightWave Server on development and test systems so may also be considered Administrators.
- Others - Anyone who is neither an Administrator or Developer but has general interest in the product.

Administrators

- Read Introduction to LightWave Server.
- Install LightWave Server as described in Installing LightWave Server.
- Start LightWave Server as described in Starting and Stopping LightWave Server.

Developers

- Read Introduction to LightWave Server.
- If also acting as an Administrator, install LightWave Server™ as described in Installing LightWave Server.
- If also acting as an Administrator, start LightWave Server™ as described in Starting and Stopping LightWave Server.
- Read Getting Started in the Developer Guide.

Others

- Read section Introduction to LightWave Server.

Introduction to LightWave Server

The following sections describe concepts and features of LightWave Server™. The information is intended for LightWave Server™ developers, and those seeking a high-level view of how LightWave Server™ works. Application programming experience may be helpful but is not required.

What is LightWave Server

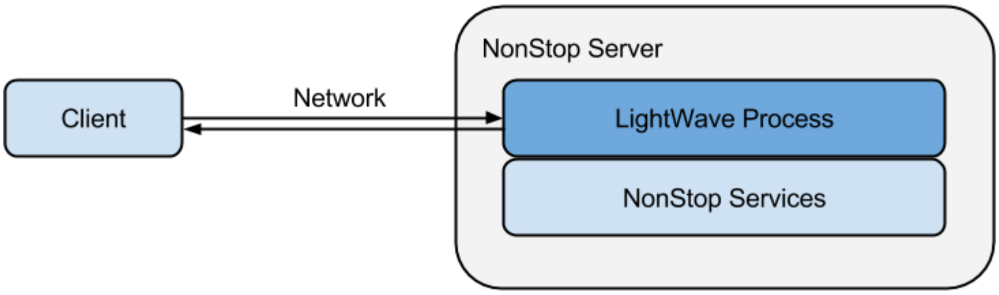
For decades, client-server middleware has been the bridge between an application's client components, which typically provide the application's user interface, and the server components, which provide the application's data and business rules. Although communications and message protocols have changed dramatically over the years, the basic concept remains the same; the application client components connect to the server components using some type of communications network, and exchange information using structured messages.

LightWave Server™ is client-server middleware that provides the bridge between client components which may be running on any platform, and server components running on HP NonStop Servers. Using a simple RESTful application programming interfaces, or *APIs*, clients can perform any of the following operations on a NonStop Server:

- Exchange a message with a standalone application process.
- Exchange a message with a Pathway application server
- Create, commit, and abort transactions and include other operations in transactions.
- Store, retrieve, or delete a file.
- Create, read, update, or delete Enscribe file records.

Client applications communicate with the LightWave Server process, which runs on HP NonStop Server under Guardian, using a TCP/IP socket connection and the HTTP protocol. Although this may sound complicated, it's the same mechanism used by web browsers to connect to web servers, and the same mechanism used by thousands of web and mobile based applications to connect to their associated back-end servers. This mechanism is simple, well understood by developers, and is used billions of times per day to connect web and mobile clients to services like Google, Facebook, and Twitter. The goal of

LightWave Server is to make it as simple to connect a client to a NonStop Server application as it is to connect a client to any of these other well known services.



LightWave Server is not specific to any particular type of NonStop application. It's an enabling technology that allows any type of client to interact with nearly any type of NonStop Server application, using communications transports and message protocols that are modern, standard, and secure. LightWave Server can be used to develop any number of applications using a NonStop Server back end, for example:

- Modernize SCOBOL "green screen" applications by developing new browser based user interfaces that use LightWave Server to communicate with the existing Pathway servers.
- Develop a mobile application for sales reps that provides order entry and status information using LightWave Server to communicate with an existing NonStop Server order processing systems.
- Provide business partners with access to existing or new NonStop Server applications through a simple RESTful API.
- Replace file upload or download applications that currently use FTP protocol with equivalent functionality using the HTTP protocol.

Invoking Service Requests

Client applications submit a request to the LightWave Server process using a specially crafted HTTP *URI*, HTTP *Method*, and HTTP *Request Body*. The HTTP *URI* indicates the API being invoked and the NonStop system resource that the request should act upon. The URI may also include additional parameter values necessary to complete the request. The HTTP *Method* indicates the operation being performed on the resource. The optional HTTP *Request Body* and *Request Headers* contain data and/or parameters necessary to complete the request. After processing the request, LightWave Server returns an HTTP *Response Status* with an optional HTTP *Response Body* and *Response Headers*. Each combination of Method, URI, and parameters is referred to as an *operation,* and groups of operations are organized into *APIs.*

User-defined APIs

User-defined APIs are a powerful feature of LightWave Server that allow you to design your own RESTful interface to your NonStop Server applications. With a user-defined API, you dictate the form of the URI and the query parameters and HTTP headers your API will use. You can map those elements to fields in the interprocess message which is sent to your application server. Likewise, you can define how replies from the server are mapped to the response that your API client receives. User-defined APIs allow you to hide the details of your server interface from users of your API.

Native APIs

LightWave Server also support a set of predefined *native* APIs that provide raw access to underlying NonStop Kernel resource. These APIs are used in conjunction with User-defined APIs. The following native APIs are provided:

- Dialog API - Manage Pathway serverclass dialogs
- Transaction API - Begin, end, and abort TMF transactions. These transactions may also be used with other APIs to enlist those operations in a transaction.

Examples

Some examples of API access using Method and URI combinations is shown below:

Method	URI	Description
GET	/systemstatus/v1/cpu	Use a user-defined API to get CPU status information from a Pathway server.
POST	/employee	Use a user-defined API to send a request to the employee server to create a new employee. The employee information is contained in the JSON request body.
POST	/lightwave/api/v1/transaction?timeout=30	Use the Native transaction API to begin a new transaction with a 30 second autoabort timeout. A transaction ID is returned in the response and may be used in subsequent User-defined API requests to enlist that request in the transaction.

Transferring Application Data

NonStop Server application components such as Pathway servers, processes, and Enscribe files typically deal with input/output data as arrays or structures of binary data. For files, these structures are commonly referred to as *records*. For processes, these structures are referred to as i*nterprocess messages* or *IPMs*. For example, the reply IPM from a Pathway server that returns customer account balance information might look like this:

```
reply IPM data: "00112233445500010023JOHN DOE  "
                 ^^                               - result code     = 00
                   ^^^^^^^^^^                     - account number  = 1122334455
                             ^^^^^^^^             - balance         = 1000.23
                                     ^^^^^^^^^^   - customer name   = JOHN DOE
```

HTTP client applications transfer data to and from host applications in the HTTP *request / response body*. The format of the data in the body is specified using the HTTP *Content-Type* header.

JSON Content

The primary content type for LightWave Server data transfer is *JavaScript Object Notation* or JSON. The use of JSON as a web application content type has become extremely popular because it's light weight, human readable, more compact and easier to manipulate than XML, and is manipulated easily from JavaScript. There are also several high-quality API libraries that provide JSON parsers for languages other than JavaScript. These factors have combined to make JSON the de facto standard for modern web-based applications. This content type is indicated in HTTP request and response bodies using the HTTP `Content-Type` header value `application/json`.

In order to easily interact with existing NonStop applications LightWave Server provides mapping services that convert JSON formatted message data to and from the binary IPM and record formats expected by the NonStop application components. Mapping instructions are stored in LightWave Server *Dictionaries* and managed using the LightWave Server Console. Dictionary definitions and can be imported directly from NonStop DDL dictionaries if available. For example, the account balance reply IPM formatted as JSON might appear like this:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 114

{
  "resultCode": "00",
  "accountNumber": "1122334455",
  "balance" : 1000.23
  "customerName": "JOHN DOE"
}
```

Security

LightWave Server™ provides a number of built in security features to ensure that your applications and NonStop Server remain secure.

Transport Security

LightWave Server uses OpenSSL to provide transport security using the latest version of TLS. Client applications can take advantage of this feature by simply invoking API requests using the HTTPS protocol.

User Authentication

LightWave Server maintains its own User and Group identity store which is managed from the LightWave Server Management Console. API clients authenticate with LightWave Server using standard HTTP Basic or Digest authentication.

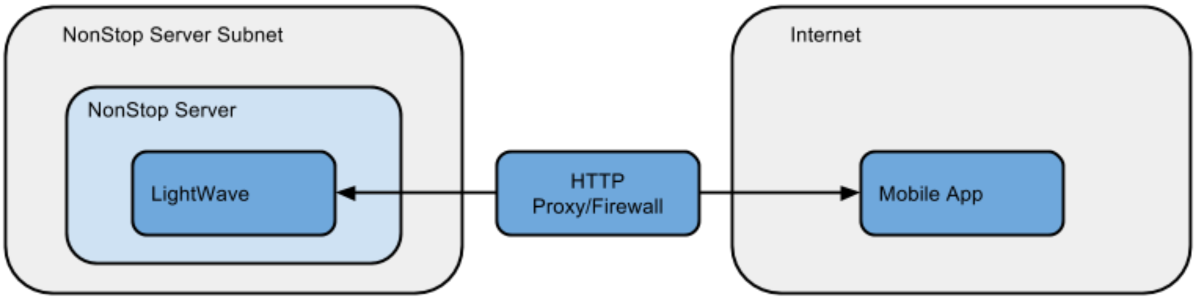
Rules Based Access Control

Access to API services is controlled using the configuration and application of Access Control Policies. Policies can be configured to restrict API service access to specific users, groups, or client source IP addresses. For example, any of the following rules can be created:

- Allow group "hr" access to the Employee service which provides the employee management API.
- Allow user "janedoe" access to the process API, but only if connecting from IP subnet 10.0.0.0/28.
- Restrict requests to specific CORS origins.

Network Isolation

Although not a specific feature of LightWave Server, because LightWave Server uses the HTTP protocol, it's possible to use any number of web-application network isolation techniques to isolate your NonStop Server network from client application networks. For example, if your NonStop Server is providing back end services for a mobile Intenet application, you can isolate your NonStop Server network from the Internet using a secure HTTP proxy. Many such web application isolation products are available providing various levels of security, access control, and logging. Please consult with your Network and/or Security personnel for more information regarding your specific needs. The diagram below shows a simple example of a proxy being used with a LightWave Server application.



Performance and Scalability

LightWave Server™ was designed to be fully scalable using the Supervisor-Worker pattern. A LightWave Server *instance* consists of a LightWave Server SERVER process pair and optionally one or more LightWave Server SWORKER processes. The SERVER process listens on the instance's configured TCP/IP ports, hosts the LightWave Server Console, and services new connection request from API clients. If no SWORKER processes have been configured, the Server process will also perform API request processing. Any number of LightWave Server instances may be running on a single system, as long as they are configured to use different TCP/IP ports.

As application load increases, SWORKER processes may be started and attached to a SERVER process. When SWORKER processes are attached, the SERVER process will delegate new connections to the SWORKER processes in round-robin fashion. Any number of SWORKER processes may be started in any CPU, providing adequate capacity for any application.

LightWave Server in Action

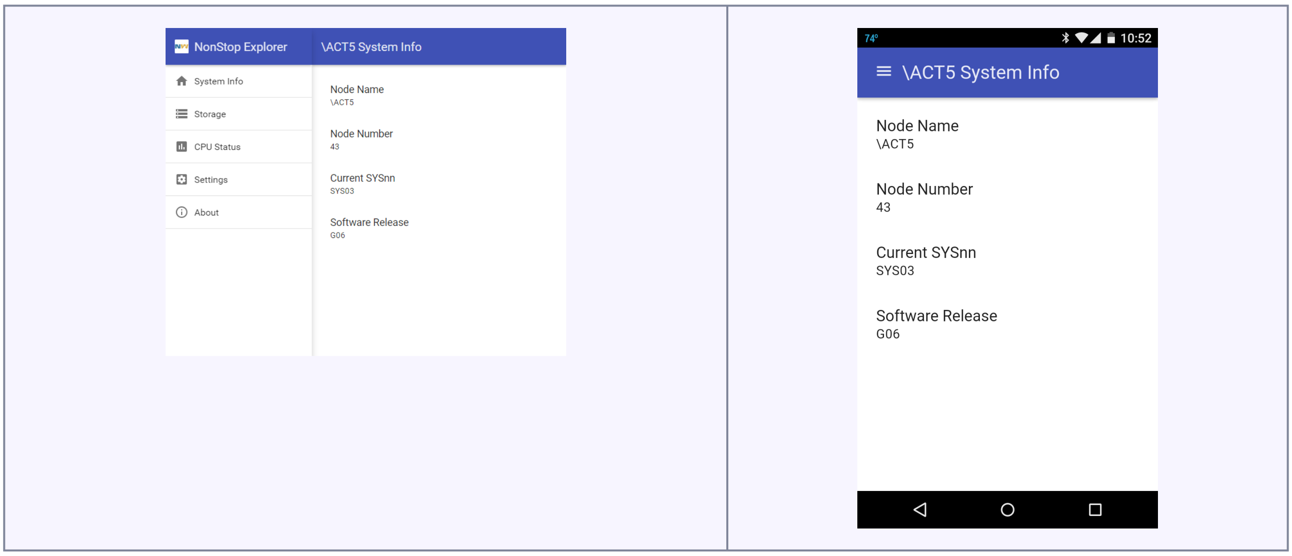
Throughout this documentation the *NonStop Explorer* sample application is used to illustrate LightWave Server™ concepts and features. NonStop Explorer allows a user to connect to a remote NonStop server, browse the Guardian file system, and view CPU utilization on the server. The application is available in both browser based and Android app versions.

It's important to understand that *NonStop Explorer* is a sample application that uses LightWave Server, not a feature of the product. The sample consists of web and Android client applications and a NonStop Server Pathway application. The client applications exchange messages with the Pathway server, over an Internet TCP/IP connection, using LightWave Server as a bridge.

The NonStop Explorer Lite sample is a simplified implementation of the NonStop Explorer application that shows the request/response messages being used by the web and app version of NonStop Explorer.

Source code for all components of the NonStop Explorer application is available on Github.

When NonStop Explorer is launched, the System Summary panel displays information including the Node Name, Node Number, SYSnn subvol, and Software Release.



\ACT5 System Info

Node Name
\ACT5

Node Number
43

Current SYSnn
SYS03

Software Release
G06

\ACT5 System Info

Node Name
\ACT5

Node Number
43

Current SYSnn
SYS03

Software Release
G06

This information is retrieved from the NonStop Explorer Pathway server via LightWave Server. The request/response exchange travels from the browser or Android device, over an Internet TCP/IP connection, to a LightWave Server instance running on a NonStop Server, to the NonStop Explorer Pathway server, and back. The following sequence occurs when the client requests the information:

- The NonStop Explorer application (Android or web) issues an HTTP request to the LightWave Server process running on the NonStop Server.
- The LightWave Server process, running on the NonStop, converts the request into an interprocess message (IPM) and sends it to the NonStop Explorer Pathway server.
- The Pathway server fills the IPM with system information and returns it to the LightWave Server Process.
- The LightWave Server process converts the IPM to an HTTP response and returns it to the NonStop Explorer application.
- The application displays the information.

All LightWave Server requests follow this same basic pattern, whether the client is communicating with an application server, managing a transaction, or reading a record from a file. How the client application submits the HTTP request to LightWave Server depends on the application platform, i.e., an application written in JavaScript will submit the application differently than one written in Java, but the basic request pattern is the same for all applications.

Please feel free to try the sample application, review the source code on GitHub, or even experiment with developing your own LightWave Server client using the NonStop Explorer Pathway Application.

Developer Guide

The Developer Guide contains the information needed to develop LightWave Server™ applications. If you're looking for information on how to install or manage LightWave Server, refer to the Administrator Guide. Before reading the Developer Guide it is recommended that you read the Introduction to LightWave Server.

Getting Started

This section provides an overview of how to get starting with developing a LightWave Server™ application. If you have not done so already, please read the Introduction to LightWave Server section before continuing.

- Read section Working with Dictionaries. Dictionaries are derived from DDL and contain the interprocess message type defintions needed by LightWave Server to map JSON to IPM and vice-versa.
- User-defined APIs are for exposing a NonStop Server application as a service. Read section Working with User-Defined APIs
- If your application will require transactions, read Working with Transactions.
- Define your application interprocess messages or file records. If you're using an existing server or file, this step may already be completed.
- When you're ready to make your application available to clients, deploy it as a service. See Deploying APIs as Services.
- Build your client application.

Tip

Visit the NuWave GitHub site for sample applications.

Working with Dictionaries

Dictionaries contain mappings between JSON message objects and binary IPMs expected by NonStop applications. The mappings enable LightWave Server™ to automatically convert alphanumeric values in JSON requests to their appropriate integer, decimal, floating point and string (with appropriate character set encoding) NonStop equivalents. Likewise, responses are automatically converted from NonStop data types to JSON format. Dictionary type definitions are referenced in the API editor when defining mappings between JSON payloads and NonStop Server application messages.

Data type definitions are stored in LightWave Server's Dictionary Repository. The LightWave Server Console allows application developers to manage the dictionary definitions. If NonStop DDL dictionaries already exist for a given NonStop application, the dictionary definitions may be imported directly into a LightWave Server dictionary.

The following sections describe the structure of the LightWave Server dictionary repository and how to manage application dictionaries.

- The Dictionary Repository
- Managing Dictionaries
- Dictionary File Reference

The Dictionary Repository

LightWave Server™ dictionaries are XML files that describe the data structure of the application IPM or record. Dictionary files are stored in the LIghtWave Server filesystem. A sample dictionary file containing the NonStop Explorer System Status request is shown below:

```
<dictionary xmlns="http://schemas.nuwavetech.com/schemas/lightwave/
dictionary-1.0.xsd">
  <types>
    <type name="GetSystemInfo" size="2" ddlName="GET-SYSTEM-INFO"
ddlModified="2014-09-09T08:55"
ddlOrigin="MXxJRHxHRVQtU1lTVEVNLUlORk98XEFDVDUuJERFTU8uTlNF">
      <element name="requestCode" type="unsignedShort" offset="0" size="2" value="5"/
>
    </type>
  </types>
```

```
</dictionary>
```

Dictionary files may be generated directly from DDL dictionaries, manually authored, or a combination of both.

Dictionary Names

In order to avoid dictionary name collisions, a convention has been established for dictionary naming. It is strongly recommended that this naming convention be followed. Dictionary names begin with the reverse internet domain name of the organization that created them, followed by the application they were created for, followed by additional dot separated qualifiers to further describe the dictionary. For example:

- com.nuwavetech.lightwave.nonstopexplorer
- com.nuwavetech.lightwave.fedwire.fedwirev1

Managing Dictionaries

LightWave Dictionaries are managed using the Server Console. See Dictionaries in the Administrator Guide.

Dictionary File Reference

The Dictionary File is an instance of an XML document that is stored in LightWave Server's filesystem. The Dictionary File contains the type definitions that LightWave Server needs to:

- map incoming JSON requests to the appropriate IPM structure for process or serverclass requests.
- map the process or serverclass reply IPM to the outgoing JSON reply.

The Dictionary Manager is used to maintain the Dictionary File, but from time to time it may be necessary to manually edit the Dictionary File. This section describes the Dictionary File structure.

- Data Types
- Annotating DDL Elements
- Schema Reference

Data Types

LightWave Supports supports the following data type names which may be specified in the *type* attribute of a dictionary file *element*.

type attribute value	Default size
int	4
short	2
long	4
longlong	8
byte	1
unsignedInt	4
unsignedShort	2
unsignedLong	4
unsignedLongLong	8
unsignedByte	1
float	4
double	8
string	no default
numeric	18
numericSignTrailing	18
numericSignEmbedded	18

type attribute value	Default size
numericSignTrailingEmbedded	18
unsignedNumeric	18
base64Binary	no default
hexBinary	no default
boolean	4

Annotating DDL Elements

The LightWave Server™ Dictionary Manager translates your DDL 'definitions' to <type> and <element> elements in your Dictionary File. However, the <element> element supports features not available in DDL. Though you could modify the generated <element> elements to take advantage of these features, you'd have to repeat the modifications if you refresh your Dictionary. Instead, you can create 'annotations' on DDL definition elements that Dictionary Manager will recognize and incorporate into your Dictionary automatically.

A LightWave annotation is merely a DDL comment on an element in a particular form:

```
* @LightWaveAttribute(attribute-name="attribute-value"[,attribute-name="attribute-value"]...)
```

where "*" in column one is the DDL comment indicator character; attribute-name and attribute-value are any of the attribute name-value pairs that are valid for the <element> element. Any number of attributes can be specified as a comma-separated list of name-value pairs. The values of the specified attributes supersede any that Dictionary Manager would ordinarily generate.

LightWave annotations can coexist with any other DDL comments you may have on a given DDL element. You must, however, include the "?COMMENTS" DDL directive in your DDL source in order to direct the DDL compiler save the comments in the compiled DDL dictionary; otherwise the comments are discarded during compilation and won't be available for the Dictionary Manager to read.

Consider the following DDL definition.

```
?COMMENTS
```

```
DEF MY-IPM.
  02 REPLY-CODE TYPE BINARY 16.
* @LightWaveAttribute(type="boolean")
  02 BOOLEAN-VALUE TYPE BINARY 32.
END.
```

Assume that the DDL element BOOLEAN-VALUE is a 32-bit integer value 0 or 1. We'd like to have LightWave Server interpret this field as a JSON boolean when serialization and deserialzation occurs. LightWave Server supports this conversion; we need only to indicate that BOOLEAN-VALUE should be treated as a 'boolean' data type. Ordinarily, the Dictionary Manager would translate this element as an 32-bit integer data type. If we annotate the DDL as shown above, the following is generated in the Dictionary File:

```
<type name="my_ipm" size="10">
  <element name="replyCode" type="short" offset="0" size="2"/>
  <element name="boolanValue" type="boolean" offset="2" size="4"/>
</type>
```

Here's another example:

```
?COMMENTS
DEF MY-IPM.
  02 REPLY-CODE TYPE BINARY 16.
  02 ARRAY-1-COUNT PIC 9(4) COMP.
* @LightWaveAttribute(dependsOn="array1Count",minOccurs="0")
  02 ARRAY-1-ITEM PIC 9(4) COMP OCCURS 20 TIMES.
  02 ARRAY-2-COUNT PIC 9(4) COMP.
  02 ARRAY-2-ITEM PIC 9(4) COMP OCCURS 0 TO 10 TIMES DEPENDING ON ARRAY-2-COUNT.
END.
```

Here our definition contains two arrays. DDL only will allow the "DEPENDING ON" clause to be used on the last element of the definition, but our first array is, logically, a variable-length array as well. We've annotated the DDL to specify the "dependsOn" element name and the "minOccurs" count. The maximum size of the array will still be allocated for the server interprocess message, but only the actual count of array items will be serialized for the Web service client. The resulting <type> is shown below:

```
<type name="my_ipm" size="">
  <element name="replyCode" type="short" offset="0" size="2"/>
  <element name="array1Count" type="unsignedShort" offset="2" size="2"/>
  <element name="array1Item" type="unsignedShort" minOccurs="0" maxOccurs="20"
offset="4" size="2" dependsOn="array_1_count"/>
  <element name="array2Count" type="unsignedShort" offset="44" size="2"/>
  <element name="array2Item" type="unsignedShort" minOccurs="0" maxOccurs="10"
offset="46" size="2" dependsOn="array_2_count"/>
```

```
</type>
```

Schema Reference

- <dictionary>
- <element>
- <type>
- <types>

<dictionary>

This is the root element of the Dictionary File.

parents	none; this is the root element
children	<types>[1]

Attributes

Attribute	Data Type	Usage	Default Value	Description
name	xsd:NCName	required	none	The dictionary name.
id	xsd:NCName	required	none	The id of the dictionary.

Example

```
<dictionary xmlns="http://schemas.nuwavetech.com/schemas/lightwave/
dictionary-1.0.xsd" name="io.nuwavetech.nonstopexplorer"
id="0749edc07692791d77800101000000d650">
...
</dictionary>
```

<element>

This element represents an element of a <type>.

parents	<type>
children	none

Attributes

Attribute	Data Type	Usage	Default Value	Description
name	xsd:NCName	required	none	The name of the element. It must be unique within the parent <type>.
offset	xsd:nonNegativeInteger	required	none	The position of this element relative to the start of the containing <type>, in bytes.
size	xsd:positiveInteger	required	none	The storage size for the element, in bytes.
type	xsd:NCName	required	none	The data type of the element. Must be one of the build-in types or the name of a <type> element.
dependsOn	xsd:NCName	optional	none	If the element represents an array, the name of the element (contained within the same <type> as this element) that indicates the actual number of array occurrences at runtime.
isSet	xsd:NCName	optional	none	References an integer element that indicates whether of not the source element should be present in the request, or was present in the response. See Working with Optional Fields

Attribute	Data Type	Usage	Default Value	Description
minOccurs	xsd:positiveInteger	optional	1	If the element represents an array, the minimum number of occurrences. If the element represents a primitive type, indicates that the element is optional in the message.
maxOccurs	xsd:positiveInteger	optional	1	If the element represents an array, the maximum number of occurrences.
stringPadding	xsd:token	optional	zeros	Controls the way elements of type string are treated. The default value is inherited from the containing <type> or the <server> whose request or reply type contains this element. Valid values are "zeros" and "spaces".
hide	xsd:integer	optional	0	Indicates whether or not the element will appear in the JSON message. Valid values are "0" or "1". A value of "1" will cause the element to be omitted from the JSON. Although the default value of the attribute is "0", this default can be altered by the private attribute of this element's parent <type> element.

Remarks

Refer to the section Advanced Topics for more information on these attributes.

Example

```
<type name="GetCpuList" size="18" ddlName="GET-CPU-LIST"
ddlModified="2014-09-09T08:55"
ddlOrigin="MXxJRHxHRVQtQ1BVLUxJU1R8XEFDVDUuJERFTU8uTlNF">
    <element name="requestCode" type="unsignedShort" offset="0" size="2" value="6"/>
    <element name="node" type="string" offset="2" size="16"/>
</type>
<type name="GetCpuListResult" size="340" ddlName="GET-CPU-LIST-RESULT"
ddlModified="2014-09-09T08:55"
ddlOrigin="MXxJRHxHRVQtQ1BVLUxJU1QtUkVTVUxUfFxBQ1Q1LiRERU1PLk5TRQ==">
```

```
    <element name="resultCode" type="unsignedShort" offset="0" size="2"/>
    <element name="node" type="string" offset="2" size="16"/>
    <element name="cpuList" type="CpuList" offset="18" size="322"/>
</type>
```

<type>

This element describes a "type". It is a container for one or more <element> elements.

parents	<types>
children	<element>[1:∞]

Attributes

Attribute	Data Type	Usage	Default Value	Description
name	xsd:NCName	required	none	The type name. Must be unique within <types>.
size	xsd:positiveInteger	required	none	The size of the type in bytes.
ddlObject	xsd:token	optional	none	Reserved for use by the Service Definition Wizard. Do not modify the contents of this attribute.
stringPadding	xsd:token	optional	zeros	Controls the way <element type="string" ...> are treated. Sets the default for <element> members of this <type> used by this server. Valid values are "zeros" and "spaces". See Remarks for the <element> element for more information.

Attribute	Data Type	Usage	Default Value	Description
encoding	xsd:NCName	optional		Indicates the default character encoding that should be used when serializing/deserializing child elements to/from the SOAP envelope. If omitted, the default encoding is the server's default encoding which is typically ISO-8859-1. This value may be altered by Server startup options. A list of available encoding names can be found in Character Encoding Names.

Example

```
<types>
   <type name="MyRequestMessage" size="14">
       <element name="RequestCode" type="short" offset="0" size="2"/>
       <element name="MyDataItem" type="string" offset="2" size="12"/>
   </type>
   <type name="MyReplyMessage" size="10">
       <element name="ReplyCode" type="short" offset="0" size="2"/>
       <element name="MyDataItem" type="double" offset="2" size="8"/>
   </type>
</types>
```

<types>

This element is a container for one or more <type> elements.

parents	<dictionary>
children	<type>[1:∞]

Example

```
<types>
   <type name="MyRequestMessage" size="14">
       <element name="RequestCode" type="short" offset="0" size="2"/>
       <element name="MyDataItem" type="string" offset="2" size="12"/>
```

```
    </type>
    <type name="MyReplyMessage" size="10">
        <element name="ReplyCode" type="short" offset="0" size="2"/>
        <element name="MyDataItem" type="double" offset="2" size="8"/>
    </type>
</types>
```

Working with User-Defined APIs

Overview

User-defined APIs are a powerful feature of LightWave Server™ that allow you to design a RESTful interface to your server. With a user-defined API, you dictate the form of the URI and the query parameters and HTTP headers your API will use. You can map those elements to fields in the interprocess message which is sent to your server. Likewise, you can define how replies from the server are mapped to the response that your API client receives. User-defined APIs allow you to hide the details of your server interface from users of your API.

APIs consist of one or more "paths" – the URIs clients will use to call your API. Each path can have one or more "methods", which correspond to the HTTP methods (verbs) GET, POST, PUT and DELETE. Each method defines a server interaction tied to a specific request message and corresponding set of replies. You specify the name of your server, the dictionary definitions that describe the request message (IPM) your server expects and one or more replies it may return. Fields in these messages can be mapped to parts of the URI path, query parameters, HTTP headers or set to specific alphanumeric values.

APIs are created using the LightWave Console to perform the following steps:

1. Specify a name and brief description.
2. Add a path.
3. Add a method for the path.
4. Specify the server name, request message and reply message(s).

Detailed instructions follow. Once you've defined your API, you must deploy your API as a service it before clients can call it.

RESTful API Design Pattern

In RESTful APIs, a path corresponds to a "resource" or a collection of resources. Example resources are employee, customer, account, order, report, etc. A server that supports CRUD operations for a given resource will typically define two paths with the following methods:

/my-api/v1/employee

GET - Gets a list of employees (the collection of employee resources)

POST - **C**reates a new employee whose employee-id is assigned by the server (adds a new employee to the 'employee' collection)

/my-api/v1/employee/{employee-id}

GET - **R**eads details about the employee with id {employee-id}

PUT - **U**pdates details about the employee with id {employee-id}

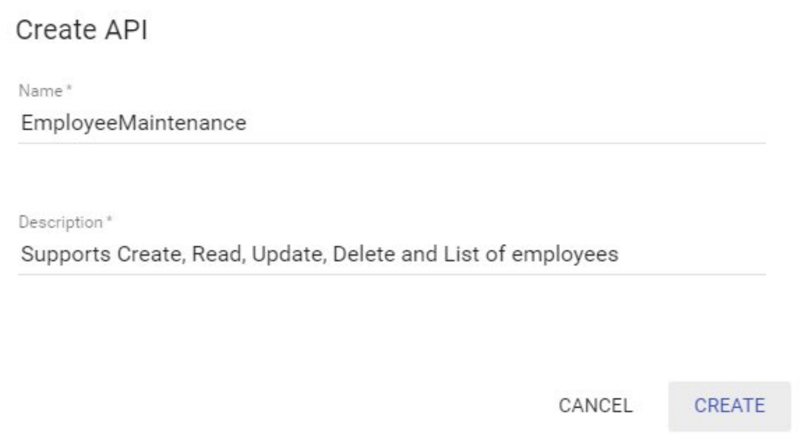
DELETE - **D**eletes the employee with id {employee-id}

The GET method is used when requesting (information about) a resource or a collection of resources. POST is used to create a new resource instance without an existing resource id, and PUT is used when creating or updating a resource using a existing resource id. Finally, DELETE is used to indicate that an existing resource should be deleted. Most other operations would use the POST method. In this example, {employee-id} is a *path parameter* – a placeholder for a value that will be supplied by the caller.

> ℹ You can only define one method of each type (GET, POST, PUT or DELETE) per path.

Create an API

You use the LightWave Console to create APIs. Select "APIs" from the console menu, which opens the API list view. Click the + icon to define a new API. Enter a name and a brief description, then press Create. The API name can be from 1 to 64 characters long and contain only letters, digits, underscores and hyphens – no spaces. The name must start with a letter.



Create API

Name *

EmployeeMaintenance

Description *

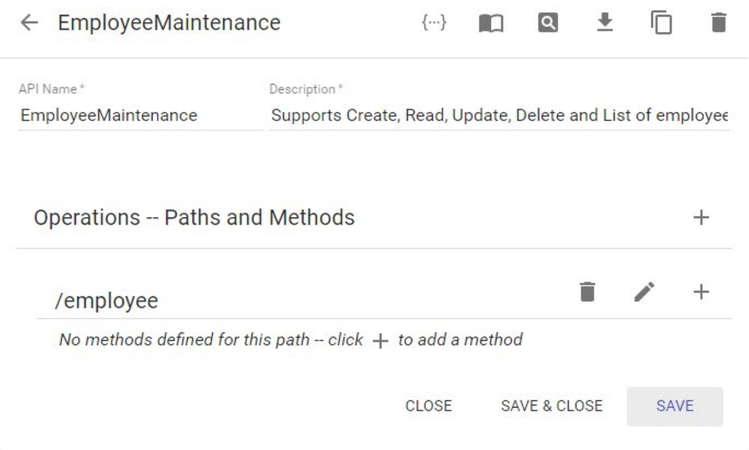
Supports Create, Read, Update, Delete and List of employees

Add a Path

Click the '+' icon on the "Paths and Methods" toolbar to add a new path. An API path must start with '/', followed by one or more path elements separated by '/'. Paths may include parameter markers, which are used to map parts of the path to fields in the server request message. Examples of paths include:

- /employee
- /employee/{employee-id}
- /my-api/v1/my-subsystem/{subsystem-id}/component/{component-id}

Parameter markers are designated by a name enclosed in curly braces '{ }'. Parameter names must start with a letter and may contain letters, digits and hyphens.



API Name * EmployeeMaintenance

Description * Supports Create, Read, Update, Delete and List of employee

/employee

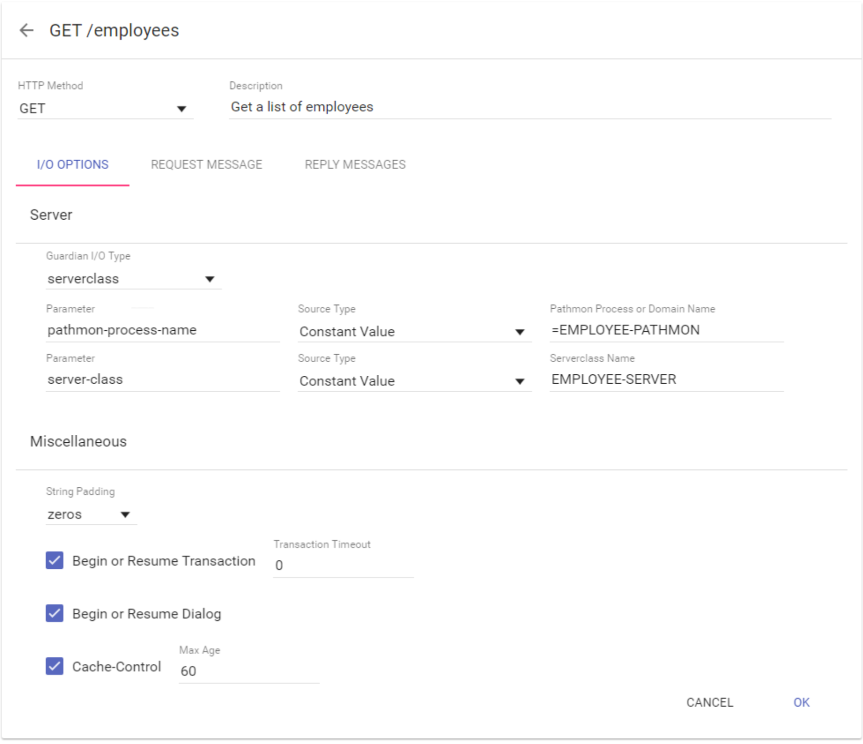
No methods defined for this path -- click + to add a method

Path specifications do not include "query parameters", such as 'details' in the URI / employee/{employee-id}?details=1. Rather, query parameters are part of a method's request message field mappings.

Add a Method for a Path

To add a method for a path, click the '+' icon to the far right of the path.

In the method detail view, select the HTTP verb that clients will use to call this method: GET, POST, PUT or DELETE, and enter a brief description.



GET /employees

HTTP Method: GET

Description: Get a list of employees

Server

Guardian I/O Type: serverclass

Parameter: pathmon-process-name | Source Type: Constant Value | Pathmon Process or Domain Name: =EMPLOYEE-PATHMON

Parameter: server-class | Source Type: Constant Value | Serverclass Name: EMPLOYEE-SERVER

Miscellaneous

String Padding: zeros

☑ Begin or Resume Transaction | Transaction Timeout: 0

☑ Begin or Resume Dialog

☑ Cache-Control | Max Age: 60

I/O Options

Select the Guardian I/O Type that LightWave should use to send messages to your server: "process", if your server runs as a standalone named server process, or "serverclass" if your server is configured as a Pathway serverclass.

Specify the name of your server here. If the Guardian I/O Type specified is "process", a process name must be supplied. The name may be a process name such as "$MYSVR" or a class MAP =DEFINE that is set to a process name. The actual name can come from one of several sources that are evaluated at time of the request.

> The maximum interprocess message size supported by LightWave Server is the same as the maximum message size supported by the Guardian message system for process servers, and NonStop TS/MP Pathsend for Pathway serverclasses. The maximum message size is typically 2MB (2097152 bytes) on modern NonStop systems, but may vary depending on NonStop software versions and the application language in use.

Source Type	You enter / select
Constant Value	a string such as $MYSVR or =MYSERVER
HTTP Request Header	the name of the HTTP header that will contain the name or =define name for the server, such as "my-server-name"
URI Path Component	the path variable you previously included in the path for this method. If the path is /{my-api}/employee/{employee-id}, you would select "{my-server}"
URI Query Parameter	the name of a query parameter that will be supplied by the caller and which will contain the name of the server. If the request URI is /employee/{employee-id}?my-server=$MYSVR, you would enter "my-server" as the name.

When Source Type is Constant Value, the client does not control the destination of the message, which is usually the desired case.

If the Guardian I/O Type specified is "serverclass", a PATHMON process name or =DEFINE name must be specified. The options are the same as described above for named processes. In addition, a serverclass name is required. As with the process name, several sources for the value of the name are available.

Miscellaneous

String padding indicates how LightWave should treat character strings in the server's request and reply messages. When set to "zeros", LightWave will pad any remaining space in the field with binary zeros when copying values to request message fields. Likewise, LightWave will expect string values in reply messages from the server to be zero-terminated (or 'null' terminated). If the server is written in the C programming language, "zeros" is usually the correct option. When set to "spaces", LightWave will pad any remaining space in character fields to spaces. This is usually the correct option if the server is written in the COBOL programming language.

If your server requires an active TMF transaction, select the 'Begin or Resume Transaction' checkbox. If checked, LightWave will begin a new transaction or resume an existing transaction prior to sending the request message to the server. Upon reply, LightWave will commit, abort or suspend the transaction according to criteria you specify. Specify a timeout, in seconds, that should be used when beginning a new transaction. A value of 0 indicates that the default system timeout should be used. Client applications indicate a transaction to be resumed by including the "lw-transaction-id" HTTP header (returned from a prior API response) in an API request.

If your server I/O Type is "serverclass" and you want LightWave to use the Pathway "dialog" protocol when communicating with your server, select the "Begin or Resume Dialog" checkbox. If checked, LightWave will begin a new Pathway dialog or continue an existing dialog when sending the request message to the server. Upon reply, LightWave will continue or abort the dialog according to criteria you specify. Client applications indicate a dialog to be continued by including the "lw-dialog-id" HTTP header (returned from a prior API response) in an API request.

if the selected HTTP method is "GET", you can enable Cache Control. This option controls the "cache-control" HTTP header returned by LightWave with the GET response. You can configure the 'max-age' value for the cache-control header, which tells browsers and proxies that they may cache (and reuse) the response for max-age seconds since the time of original response, thus reducing the load on the server from repeated requests for the same information.

Request Message

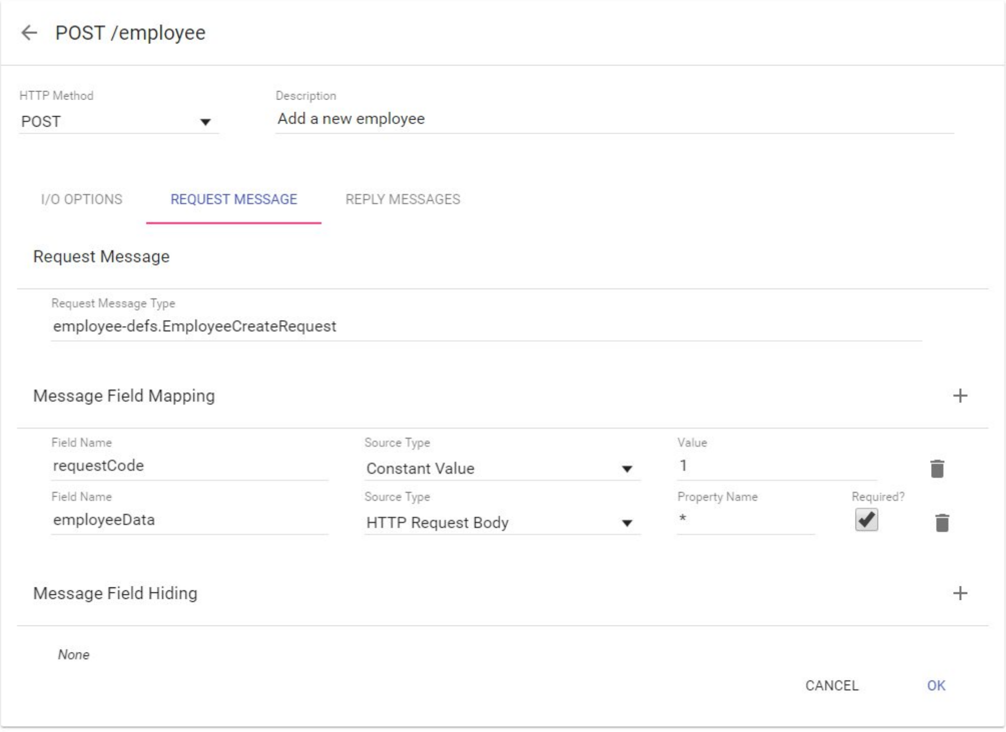
You must specify the name of the dictionary definition that describes your server's request message. Click the field to select from a list of existing dictionary definitions.

Message Field Mapping

Field mapping allows you to map values from various sources to fields of your server's request message. For each mapping, the value of the source in the HTTP request will be mapped to the specified field in the request message.

Source Type	You enter / select
Constant Value	a string, numeric value, or substitution variable
URI Path Component	a {path-variable} you specified in the path for this method
URI Query Parameter	the name of a query parameter that will be supplied in the request URI
HTTP Request Body	the name element that will be supplied in the request body, or "*" for the entire request body

<table>
<tr><th>Source Type</th><th>You enter / select</th></tr>
<tr><td>HTTP Request Header</td><td>the name of the HTTP header supplied with the request</td></tr>
<tr><td>HTTP Request Element</td><td>The name of an element of the HTTP request:
<table>
<tr><td>URI</td><td>The URI of the request, e.g., /employee/8213912?history=true</td></tr>
<tr><td>method</td><td>The HTTP method (verb) of the request, e.g., GET, PUT, ...</td></tr>
<tr><td>username</td><td>The authenticated username of the caller</td></tr>
</table></td></tr>
<tr><td>HTTP Form Field</td><td>when the request body is 'form-encoded', the name of a form field</td></tr>
</table>



POST /employee

HTTP Method: POST

Description: Add a new employee

Request Message

Request Message Type: employee-defs.EmployeeCreateRequest

Message Field Mapping

Field Name	Source Type	Value / Property Name	Required?
requestCode	Constant Value	1	
employeeData	HTTP Request Body	*	☑

Message Field Hiding

None

Message field mapping is often used to set the 'request code' field to a specific value, as in the example above. The entire request message, or a subset of the message, can be mapped from JSON content in the body of the HTTP request. Using these techniques, you can hide the details of the server interface from the client. Use "*" for the field name to indicate the entire request message should be mapped from the HTTP request body, otherwise specify the name of a field that is to be mapped.

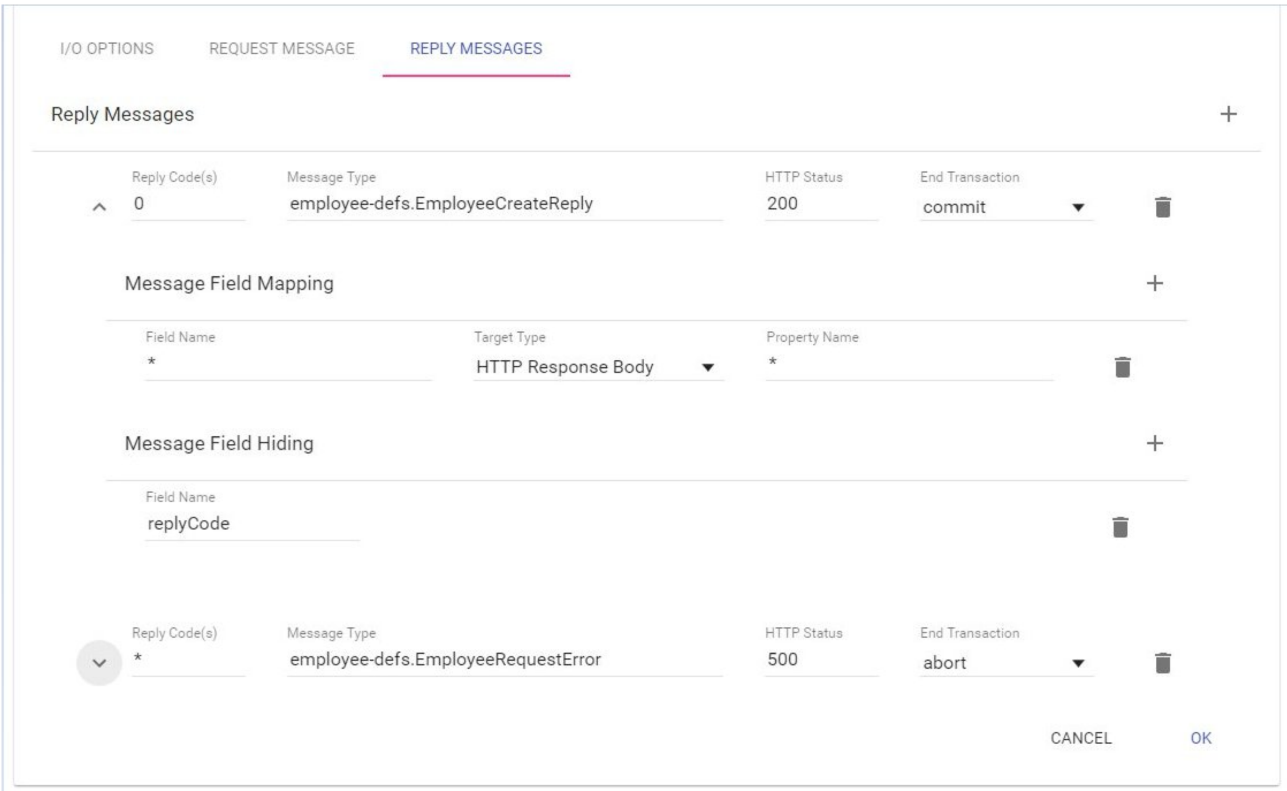
Message field mappings of some Source Types may be specified as 'required' by checking the Required checkbox. If a required mapping is not supplied in the request by the client, LightWave Server will not send the request to the server and will return an error to the client.

Message Field Hiding

Message field hiding is another technique that may be used to hide details of your server from clients. LightWave will not map any client supplied data to hidden fields, even if they are contained within another mapped field. Depending on the structure of your server request message, it may be easier to map only specific fields from particular client sources, or to map the entire request message from the HTTP Request body, but hide certain other fields, such as the 'request code'.

Reply Messages

Each server request results in a reply. The reply message may have a different structure depending on the outcome of the request (e.g. success vs. failure). You should define a reply message for each type of reply message type that the server may return. Click the ⌄ icon found to the left of the Reply Code field to reveal the Message Field Mapping and Message Field Hiding configuration for the reply.



Reply Messages

Reply Code(s)	Message Type	HTTP Status	End Transaction
0	employee-defs.EmployeeCreateReply	200	commit

Message Field Mapping

Field Name	Target Type	Property Name
*	HTTP Response Body	*

Message Field Hiding

Field Name
replyCode

Reply Code(s)	Message Type	HTTP Status	End Transaction
*	employee-defs.EmployeeRequestError	500	abort

Reply Code(s)

Specify the reply code (integer value contained in the first two bytes of the server reply) or codes that yield a particular message type. You may supply a range as 'low:high' or a comma-separated list of values and ranges. You may also specify '*', which equates to the set of all reply codes not otherwise specified.

Message Type

The dictionary message type associated with the reply code(s) you specified. To specify the reply message type, click the field to select the appropriate dictionary definition.

HTTP Status

Every response must include a 3-digit HTTP status code. The status code is used by client applications to determine if the request succeeded, and hence the expected content of the response. Therefore, you may not use the same HTTP status code for more than one reply of a particular request.

✓ Tip: if you have more than one "success" response, consider defining a separate method (on a separate path, perhaps) that returns the other success response(s). Alternatively, you can use a different 'success' HTTP status code for each success response, e.g. 200, 240, 250, etc.

The HTTP standard (RFC 7231) defines a number of status codes. Generally, 2xx status codes indicate 'success', 4xx status codes indicate 'client error' and 5xx status codes indicate 'server error'. Although you can specify any 3-digit value for the HTTP status code for a response, it is recommended that you use only the following codes since other codes may be interpreted by any intermediate HTTP proxies, messaging frameworks or components (e.g. XMLHttpRequest), etc. to have a meaning you do not intend and may alter message processing:

200 OK
201 Created
202 Accepted
204 No Content
240-290 (success, user defined)
404 Not Found
460-490 (client error, user defined)
520-590 (server error, user defined)

According to the RFC, undefined (i.e., user defined) status codes should be treated by intermediaries as 'x00' (e.g., 460 treated as 400).

Transaction State

If the 'Begin or Resume Transaction' I/O Option is selected for the method being defined, a "Transaction Sate" field is enabled for each reply. This field indicates the desired disposition of the TMF transaction upon request completion depending on the reply code. LightWave will automatically 'commit', 'abort' or 'suspend' the transaction accordingly, and return the following HTTP response headers to the client:

HTTP header name	Description
lw-transaction-id	the transaction id
lw-transaction-state	the state of the transaction: committed, aborted, suspended or unknown

When the transaction state is 'suspended', the transaction may be used again by including the returned "lw_transaction_id" HTTP header value in subsequent requests; otherwise, it must be explicitly ended using the native Transaction API.

Dialog State

If the 'Begin or Resume Dialog' I/O Option is selected for the method being defined, a "Dialog State" field is enabled for each reply. This field indicates the desired disposition of the Pathway dialog upon request completion depending on the reply code. LightWave will automatically 'continue' or 'abort' the dialog accordingly, and return the following HTTP response headers to the client:

HTTP header name	Description
lw-dialog-id	the dialog id
lw-dialog-state	the state of the dialog: ended, continued, aborted, or unknown

When the dialog state is 'continued', the dialog may be continued by including the returned "lw_dialog_id" HTTP header value in a subsequent request; otherwise, it must be explicitly aborted using the native Dialog API. Note that it is actually the Pathway server that decides whether a dialog will continue or not – when you specify the "continue" option, LightWave reports (via the lw-dialog-state response header) the server's decision.

Message Field Mapping

As with the request message, reply message fields can be mapped to elements of the HTTP response. However, mapping targets are limited to HTTP headers and body, since the other mapping types are not applicable to HTTP responses. If your server is returning data to the client, it must be mapped in some way. A mapping of field name '*' to HTTP Response Body property name "*" will map the entire server reply to the HTTP reply body.

Message Field Hiding

As with the request message, reply message fields can be hidden. A hidden field will not appear in the response.

Using Substitution Variables

When a message field mapping source type is Constant, a substitution variable may be supplied as the constant value. The field will be set with the corresponding value of the substitution variable. The following variables are available:

Variable name	Value set in the corresponding field
${lw.juliantimestamp}	The current julian timestamp value. The target field must be an 8 byte integer.
${lw.julianday}	The current julian day value. The target field must be a 4 byte integer.
${lw.timestamp48}	The current 48 bit timestamp. The target field must be a 6 byte string.

Working with Native APIs

LightWave Server™ supports a set of predefined *native* APIs that provide raw access to underlying NonStop Kernel resource. Native APIs provide access to operations in addition to those provided by customer user defined APIs such as transaction and Pathway dialog control.

- Invoking Native Requests
- Native API Reference

Invoking Native Requests

Client applications submit a request to the LightWave Server™ process using a specially crafted HTTP URI, HTTP method, and HTTP request headers and body. The HTTP URI indicates the API being invoked and the NonStop system resource that the request should act upon. The HTTP method indicates the operation being performed on the resource. The HTTP headers and request body contains data and/or parameters necessary to complete the request. After processing the request, LightWave Server returns an HTTP response with headers and response body containing the results of the request.

Request Method Selection

The HTTP method used for each request determines the operation to perform on the resource. In general, the method correlates with the resource operation as follows:

Method	Operation
POST	Change the state of the resource. This method is used when creating or ending a transaction.

Method	Operation
DELETE	Delete the resource. This method is used when aborting a transaction or aborting a Pathway dialog.

Request URI Construction

The Request URI uses the following general format:

```
/lightwave/api/{version}/{api-name}/{resource-parameters}
```

Path element	Description
version	The version of the API being invoked. Currently this value must be "v1" for all APIs.
api-name	The name of the API being invoked, which must be *dialog* or *transaction*.
resource-parameters	The resource-parameters value varies depending on the API being invoked. These parameters are documentation in the Native API Reference.

Examples

Some examples of API access using Method and URI combinations is shown below:

Method	URI	Description
POST	/lightwave/api/v1/transaction?timeout=30	Create a new transaction with a 30 second autoabort timeout. A transaction ID is returned in the response.
POST	/lightwave/api/v1/transaction/018968f900002a72f33b2b78a99b2536	Commit the transaction with the transaction ID 018968f900002a72f33b2b78a99b2536

Method	URI	Description
DELETE	/lightwave/api/v1/dialog/ 0100000000000000	Abort a dialog.

Native API Reference

This section contains information on how to invoke the Native APIs. Throughout the API reference there are examples that illustrate HTTP requests and response. Common HTTP headers such as "Server", "Date", "Content-Length", etc. have been omitted in these examples for brevity.

API References

- Dialog API - Manage Pathway serverclass dialogs
- Transaction API - Begin, end, and abort TMF transactions. These transactions may also be used with other APIs to enlist those operations in a transaction.

Dialog API

The Dialog API allows a client application to abort an existing Pathway serverclass dialog. The following API method is available:

- Abort a Dialog

Abort a Dialog

Aborts an existing dialog. For more information on Pathway Dialog processing refer to *HP NonStop TS/MP Pathsend and Server Programming Manual*.

Path Components

```
DELETE /lightwave/api/v1/dialog/{dialog-id}
```

Name	Use	Description
dialog-id	require d	A dialog ID returned from a previous request which had the **Begin or Resume Dialog** I/O Options selected.

Parameters

Request

Name	Use	Description
lw-correlation-id	optional	A caller supplied identifier to be copied to the response.
lw-suppress-status	optional	Use to indicate that the HTTP response status code should be "200 OK", regardless of the outcome of the operation.
lw-transaction-id	optional	The transaction ID of the transaction in which this request should be enlisted

Response

Name	Description
lw-correlation-id	The value of the correlationId supplied in the request, if any.
lw-dialog-id	The Dialog ID associated with the request.
lw-dialog-state	The current statue of the Dialog.
lw-error	If an error occurred, a description of the error.
lw-http-reason	The HTTP response reason. Present if lw-suppress-status was supplied in the request.
lw-http-status	The HTTP response status. Present if lw-suppress-status was supplied in the request.

Name	Description
lw-transaction-id	The transaction ID of the TMF transaction associated with the request, if any.
lw-transaction-state	The transaction state of the TMF transaction associated with the request, if any.

Example

Request

```
DELETE http://lightwave.example.com/lightwave/api/v1/dialog/0001000000000002 HTTP/1.1
Content-length: 0
```

Response

```
HTTP/1.1 200 OK
Content-Length: 0
lw-dialog-id: 0001000000000002
lw-dialog-state: aborted
```

Transaction API

The Transaction API allows a client application to begin and subsequently end or abort a TMF transaction. Transactions may also be used with other APIs to enlist their activity in the transaction. The following API methods are available:

- Begin a Transaction
- End a Transaction
- Abort a Transaction

Begin a Transaction

Begins a new transaction and returns the transaction ID. The transaction ID value can be used in subsequent requests to enlist the request in the transaction, or to commit or abort the transaction.

Path Components

```
POST /lightwave/api/v1/transaction
```

Parameters

Request

Name	Use	Description
lw-correlation-id	optional	A caller supplied identifier to be copied to the response.
lw-suppress-status	optional	Use to indicate that the HTTP response status code should be "200 OK", regardless of the outcome of the operation.
lw-transaction-timeout	optional	Use to set the timeout in seconds for the new transaction. If not supplied or set to 0, the SERVER default timeout is used. See --*default-tx-timeout* in the SERVER command line reference.

Response

Name	Description
lw-correlation-id	The value of the correlationId supplied in the request, if any.
lw-error	If an error occurred, a description of the error.
lw-http-reason	The HTTP response reason. Present if lw-suppress-status was supplied in the request.
lw-http-status	The HTTP response status. Present if lw-suppress-status was supplied in the request.

Name	Description
lw-transaction-id	The transaction ID of the TMF transaction associated with the request, if any.
lw-transaction-state	The transaction state of the TMF transaction associated with the request, if any.

Example

Request

```
POST http://lightwave.example.com/lightwave/api/v1/transaction HTTP/1.1
Content-Length: 0
lw-transaction-timeout: 30
```

Response

```
HTTP/1.1 200 OK
Content-Length: 0
lw-transaction-state: suspended
lw-transaction-id: 2c00031c9b890000c40ff4182950cac4
```

End a Transaction

Ends (commits) a transaction.

Path Components

```
POST /lightwave/api/v1/transaction/{transaction-id}
```

Name	Use	Description
transaction-id	required	The transaction ID value returned from a previous begin transaction call.

Parameters

Request

Name	Use	Description
lw-correlation-id	optional	A caller supplied identifier to be copied to the response.
lw-suppress-status	optional	Use to indicate that the HTTP response status code should be "200 OK", regardless of the outcome of the operation.

Response

Name	Description
lw-correlation-id	The value of the correlationId supplied in the request, if any.
lw-error	If an error occurred, a description of the error.
lw-http-reason	The HTTP response reason. Present if lw-suppress-status was supplied in the request.
lw-http-status	The HTTP response status. Present if lw-suppress-status was supplied in the request.
lw-transaction-id	The transaction ID of the TMF transaction associated with the request, if any.
lw-transaction-state	The transaction state of the TMF transaction associated with the request, if any.

Example

Request

```
POST http://lightwave.example.com/lightwave/api/v1/transaction/
2c00031c9b890000c40ff4182950cac4 HTTP/1.1
Content-Length: 0
```

Response

```
HTTP/1.1 200 OK
Content-Length: 0
lw-transaction-state: committed
lw-transaction-id: 2c00031c9b890000c40ff4182950cac4
```

Abort a Transaction

Aborts the transaction indicated by the specified transaction-id.

Path Components

```
DELETE /lightwave/api/v1/transaction/{transaction-id}
```

Name	Use	Description
transaction-id	required	The transaction ID value returned from a previous begin transaction call.

Parameters

Request

Name	Use	Description
lw-correlation-id	optional	A caller supplied identifier to be copied to the response.

Name	Use	Description
lw-suppress-status	optional	Use to indicate that the HTTP response status code should be "200 OK", regardless of the outcome of the operation.

Response

Name	Description
lw-correlation-id	The value of the lw-correlation-id supplied in the request, if any.
lw-error	If an error occurred, a description of the error.
lw-http-reason	The HTTP response reason. Present if lw-suppress-status was supplied in the request.
lw-http-status	The HTTP response status. Present if lw-suppress-status was supplied in the request.
lw-transaction-id	The transaction ID of the TMF transaction associated with the request, if any.
lw-transaction-state	The transaction state of the TMF transaction associated with the request, if any.

Example

Request

```
DELETE http://lightwave.example.com/lightwave/api/v1/transaction/
2c00031c9b890000c40ff4182950cac4 HTTP/1.1
Content-Length: 0
```

Response

```
HTTP/1.1 200 OK
```

```
Content-Length: 0
lw-transaction-state: aborted
lw-transaction-id: 2c00031c9b890000c40ff4182950cac4
```

Parameter Reference

Request parameters provide additional instructions for processing the request and may be provided as query string parameters or HTTP headers. If the same parameter is provided as both a query string parameter and header, the query string parameter takes precedent.

Response parameters provide additional information in the response and are always provided as HTTP headers.

Note that all parameters are prefixed with *lw-* in order to avoid collisions with any user parameters.

- lw-correlation-id
- lw-dialog-id
- lw-dialog-state
- lw-error
- lw-http-reason
- lw-http-status
- lw-suppress-status
- lw-transaction-id
- lw-transaction-state
- lw-transaction-timeout

lw-correlation-id

An HTTP header or query string argument with the name `correlation-id` and string value. Use this parameter in a request to specify a value to be copied to the response. The correlation ID can be used to match requests with responses when requests are submitted to LightWave Server asynchronously.

Query String Example

```
http://lightwave.example.com/uri?correlation-id={a2c93b13-7443-4cdf-ae27-
ea1cea54ea75}
```

Header Example

```
POST http://lightwave.example.com/uri HTTP/1.1
```

```
correlation-id: {a2c93b13-7443-4cdf-ae27-ea1cea54ea75}
```

lw-dialog-id

A bi-directional parameter containing the Dialog ID of a Pathway Dialog. The initial value of the Dialog ID is returned in a request which has the **Begin or Resume Dialog** I/O Options selected. Subsequent requests should include this parameter to be included in the dialog.

Example

```
DELETE http://lightwave.example.com/lightwave/api/v1/dialog/0001000000000002 HTTP/1.1

HTTP/1.1 204 NO CONTENT
lw-dialog-id: 0001000000000002
lw-dialog-state: aborted
```

lw-dialog-state

A response parameter that describes the current state of the Pathway Dialog when a request begins, continues, ends, or aborts Pathway Dialog. The dialog state value may be "continued", "ended", or "aborted".

Example

```
DELETE http://lightwave.example.com/lightwave/api/v1/dialog/0001000000000002 HTTP/1.1

HTTP/1.1 204 NO CONTENT
lw-dialog-id: 0001000000000002
lw-dialog-state: aborted
```

lw-error

A group of HTTP headers returned when the request results in an error. For more information on error handling see Working with Errors

The headers include:

Name	Value
lw-error-source	A string indicating the source of the error, which may be: "lightwave" - LightWave error. "nsk" - NonStop filesystem or system procedure error. "pathway" - Pathway error.
lw-error-code	The error code.
lw-error-subcode	The error subcode.
lw-error-message	The error message.

Example

```
POST http://lightwave.example.com/lightwave/api/v1/transaction/not-a-transaction-id
HTTP/1.1
Content-Length: 0

HTTP/1.1 400 Bad Request
lw-error-source: nsk
lw-error-code: 78
lw-error-subcode: 0
lw-error-message: Transaction identifier is invalid or obsolete.
```

lw-http-reason

A response parameter returned when the request includes the lw-suppress-status parameter set to true.

Example

```
POST /lightwave/api/v1/transaction/2c00031c9b890000c40ff4182950cac4
lw-suppress-status: 1

HTTP/1.1 200 OK
lw-http-status: 404
lw-http-reason: Not Found
```

lw-http-status

A response parameter returned when the request includes the lw-suppress-status parameter set to true.

Example

```
POST /lightwave/api/v1/transaction/2c00031c9b890000c40ff4182950cac4
lw-suppress-status: 1

HTTP/1.1 200 OK
lw-http-status: 404
lw-http-reason: Not Found
```

lw-suppress-status

Use this parameter in a request to cause the HTTP response status value to be "200 OK" regardless of the outcome of the request. The actual response status and reason phrase will be returned in the lw-http-status and lw-http-reason parameters. This parameter may be required if the HTTP client framework requires a "200 OK" status in order to return the response data.

This parameter does not appear in responses. The parameter value should be a positive integer or the string "true".

An HTTP header or query string argument with the name suppress-status and non-zero numeric value indicating that response codes should be suppressed.

Query String Example

```
http://lightwave.example.com/uri?lw-suppress-status=true
```

Header Example

```
http://lightwave.example.com/uri
lw-suppress-status: 1
```

Use this parameter in a request to cause the HTTP response status value to be "200 OK" regardless of the outcome of the request. The actual response status and reason phrase will be returned in the lw-http-status and lw-http-reason parameters. This parameter may be required the HTTP client framework requires a "200 OK" status in order to return the response data.

This parameter does not appear in responses. The parameter value should be a positive integer or the string "true".

An HTTP header or query string argument with the name suppress-status and non-zero numeric value indicating that response codes should be suppressed.

Query String Example

```
http://lightwave.example.com/uri?lw-suppress-status=true
```

Header Example

```
http://lightwave.example.com/uri
lw-suppress-status: 1
```

lw-transaction-id

A bi-directional parameter containing the transaction ID of an outstanding TMF Transaction.. The initial value of the Transaction ID is returned in a request to Begin a Transaction. Subsequent requests should include the Transaction ID in order to enlist the request in the Transaction.

```
POST http://lightwave.example.com/lightwave/api/v1/transaction HTTP/1.1
lw-transaction-timeout: 30

HTTP/1.1 200 OK
Content-Type: application/json
lw-transaction-id: 2c00031c9bb70000380b806a992f7c6e
lw-transaction-state: suspended
```

lw-transaction-state

A response parameter returned by any operation that creates, updates, or is enlisted in a TMF transaction. The parameter value may be "suspended", "committed", "aborted", or "invalid".

```
POST http://lightwave.example.com/lightwave/api/v1/transaction HTTP/1.1
lw-transaction-timeout: 30

HTTP/1.1 200 OK
Content-Type: application/json
lw-transaction-id: 2c00031c9bb70000380b806a992f7c6e
```

```
lw-transaction-state: suspended
```

lw-transaction-timeout

Use this parameter in specify the number of seconds after which a new transaction will be auto-aborted by the system. If omitted or 0, the system default auto-abort timeout is used. This parameter is only effective when used with request that creates a new TMF Transaction.

```
POST http://lightwave.example.com/lightwave/api/v1/transaction HTTP/1.1
lw-transaction-timeout: 30

HTTP/1.1 200 OK
Content-Type: application/json
lw-transaction-id: 2c00031c9bb70000380b806a992f7c6e
lw-transaction-state: suspended
```

Working with Errors

HTTP Status Codes

REST API methods return three-digit HTTP "status codes" to indicate the outcome of a request. By convention, codes in the range 200-299 indicate success. Codes in the range 400-499 indicate an error in the request. Codes in the range 500-599 indicate a server-side error.

The HTTP status codes returned by LightWave Server™ when calling user-defined API methods may come from one of two sources:

- a response mapping in the user-defined API method
- from LightWave Server itself, in the case that the user-defined API method could not be executed

In the case of a response mapping, it's up to the API designer (developer) to decide which HTTP status codes and message content should be returned based upon the response code from the server application.

In the case of status codes returned by LightWave Server itself, the following may be returned:

Code	Name	Description
400-level errors are returned when there is a problem with the request. These are generally due to a client application programming error.		
400	Bad Request	The request is malformed. For example, required query parameters missing, body content missing or invalid JSON, etc. Also, a if bad transaction id is supplied, bad Pathway dialog id, etc.
401	Unauthorized	No HTTP Authentication header was supplied and the Service has an Access Control Policy that is restricted to a particular user or group.
403	Forbidden	An HTTP Authentication header was supplied, but the Service's Access Control Policy does not permit the authenticated user (due to user, group or source IP restrictions).
404	Not Found	The supplied URI (the 'path') does not match any path belonging to any deployed Service (on the TCP/IP port).
405	Method Not Allowed	The supplied URI is valid, but the requested method (GET, POST, PUT, etc.) is not defined for the URI.
500-level codes are returned for errors that are not due to the content of the request itself -- there is nothing the client application can do to fix the problem. There are typically server configuration errors.		
500	Internal Server Error	Any error that prevents LightWave from communicating with your server application (e.g., a Pathsend or Guardian error). Other causes include license expiration.
501	Not Implemented	A method other than OPTIONS, GET, POST, PUT or DELETE was attempted.

HTTP Headers

In addition to the status code, error responses generated by LightWave Server include error details within the following HTTP headers:

HTTP Header	Description
lw-error-source	The source of the error; "lightwave", "nsk", or "pathway"
lw-error-code	The error number within the 'error source'
lw-error-subcode	Additional detail for 'error code', if applicable
lw-error-text	A textual description of the error

Error responses generated by LightWave Server have no body content, i.e. content-length = 0. Error responses returned from user-defined API methods do not include 'lw-error' headers and may include body content.

Interpreting lw-error-code

<table>
<tr><th>lw-error-source</th><th>lw-error-code</th><th>lw-error-subcode</th></tr>
<tr><td>lightwave</td><td>
<table>
<tr><th>value</th><th>name</th><th>description</th></tr>
<tr><td>9013</td><td>deserialization error</td><td>Error converting the request JSON body content to server IPM format</td></tr>
<tr><td>9014</td><td>serialization error</td><td>Error converting the server response IPM content to JSON format</td></tr>
<tr><td>9016</td><td>API compiler error</td><td>Error in API definition</td></tr>
</table>
</td><td>n/a</td></tr>
</table>

lw-error-source	lw-error-code	lw-error-subcode
nsk	Guardian File System Error; see the HPE Guardian Procedure Errors and Messages Manual	n/a
pathway	Pathsend Error; see "Pathsend Errors" in the HPE NonStop TS/MP 2.5 Pathsend and Server Programming Manual	Guardian File System Error, when applicable

Suppressing HTTP Status Codes

Some client frameworks will not return response content if the HTTP Status Code is not 200. To overcome this, use the lw-suppress-status parameter when invoking the request. When this parameter is present, the response status code will be 200 regardless of the outcome of the operation. The actual status information is provided in the response lw-http-status and lw-http-reason parameters. When an error occurs, additional error detail is returned in the response lw-error parameter group.

The following examples show how the lw-suppress-status parameter alters the response for an HTTP "400 Bad Request" error returned by the Process API when the target server does not exist. In this example, if lw-suppress-status were not used, the returned HTTP status code would be 400.

Request

```
POST /lightwave/api/v1/transaction/not-a-transaction-id HTTP/1.1
lw-suppress-status: 1
Content-Length: 0
```

Response

```
HTTP/1.1 200 OK
lw-http-status: 400
lw-http-reason: Bad Request
lw-error-source: nsk
lw-error-code: 78
lw-error-subcode: 0
lw-error-message: Transaction identifier is invalid or obsolete.
```

Working with Transactions

Using the Transaction API, client applications may create then commit or abort TM/MP (TMF) transactions. When a transaction is created, an opaque transaction ID is returned. The transaction ID is used in subsequent calls to commit or abort the transaction, and may be supplied as parameters to other API requests in order to enlist that request in the transaction.

The following example shows how to create a transaction, send a serverclass request enlisted in the transaction, and commit the transaction.

Begin the Transaction

Request

```
POST http://lightwave.example.com/lightwave/api/v1/transaction HTTP/1.1
Content-Type: application/json
lw-transaction-timeout: 30
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
lw-transaction-id: 2c00031c9bb70000380b806a992f7c6e
lw-transaction-state: suspended
```

Send the Serverclass Request with the Transaction

Request

```
POST http://lightwave.example.com/lightwave/api/v1/serverclass/=ne^pathmon/nesvr
HTTP/1.1
Content-Type: application/json
lw-transaction-id: 2c00031c9bb70000380b806a992f7c6e

{
  "requestCode" : 1
}
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
lw-transaction-id: 2c00031c9bb70000380b806a992f7c6e
lw-transaction-state: suspended

{
  "resultCode": 0,
  "node": "\\ACT5",
  "systemInfo": {
    "nodeNumber": 43,
    "systemSubvol": "SYS03",
    "rvu": "G06"
  }
}
```

Commit the Transaction

Request

```
POST http://lightwave.example.com/lightwave/api/v1/transaction/
2c00031c9bb70000380b806a992f7c6e HTTP/1.1
Content-Length: 0
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
lw-transaction-id: 2c00031c9bb70000380b806a992f7c6e
lw-transaction-state: committed
```

Deploying APIs as Services

An API must be deployed as a service before you can test it, or it can be used by clients. To prepare to deploy a service:

1. Create a user-defined API or determine which native API you wish to deploy

2. Create an Access Control Policy for the service. Access Control Policies define what users can access the service and from what IP source addresses service requests can originate. An existing ACP may be used in more than one service. See Access Control Policies.

Once the API is deined and the Access Control Policy is determined, you can deploy the service.

Advanced Topics

- Array Processing
- Character String Encoding
- Character String Padding
- Sensitive Data Masking
- Working with Optional Elements
- Using MEASURE Counters
- Message Logging
- Using BLOBs
- Working with Timestamps

Array Processing

The dependsOn attribute controls the way arrays are treated when deserialization and serialization occurs. On deserialization, LightWave Server will automatically calculate a value for the 'dependsOn' element based on the number of array elements present in the request. On serialization, LightWave Server expects the dependsOn element to have been set correctly by the server. It serializes to the response only the number of array elements indicated by the dependsOn element. The element referenced by the dependsOn attribute must be an integer or numeric type.

Character String Encoding

When LightWave Server copies "string" fields between the JSON message elements and the server IPM, by default the data is copied to or from the IPM using ISO-8859-1 encoding. The encoding used can be altered on the element, the API, or set at SERVER process startup.

- To set the encoding for an individual element, set the encoding attribute on the element.
- To set the default encoding on all string elements used by an API method, set the encoding on the method in the API editor.
- To set the default encoding for all services on a specific SERVER instance set the --default-encoding startup option to the desired encoding.

For example, a request or response containing the following schema structure:

```
<type name="Request">
    <element name="string_SHIFT_JIS" type="string" offset="0" size="32"
encoding="SHIFT_JIS"/>
</type>
```

Would be serialized or deserialized using the SHIFT_JIS encoding, producing the following JSON and IPM data:

```
{
  "string_SHIFT_JIS": "ぐけげこごさざしじすずせぜそぞた"
}
```

```
0x00000000: 82ae 82af 82b0 82b1 82b2 82b3 82b4 82b5  00000000: ................
0x00000010: 82b6 82b7 82b8 82b9 82ba 82bb 82bc 82bd  00000016: ................
```

The value of the encoding attribute or the --default-encoding startup option can be any one of the Character Encoding Names values. Note that LightWave Server always uses UTF-8 encoding for the message payload. Refer to Character Encoding Names for a complete list of the available character encoding names.

Character String Padding

The stringPadding attribute controls the way elements of type "string" are treated when LightWave Server converts request input JSON string values to server interprocess request messages ("deserialization") and server reply interprocess messages to JSON response string values ("serialization"). A stringPadding value of "zeros" causes the string to be padded with zeros to the maximum field length (aka "null-terminated") in server request messages and expected to be zero-terminated (not necessarily padded) in the server reply message. A stringPadding value of "spaces" causes the string to be padded with spaces in the server request and expected to be padded with spaces in the server reply. It is usually easier to specify the stringPadding setting on the containing <type> than on every string element. However, it may be useful to set stringPadding on a small number of elements to override the default for the entire <type> which applies to the remainder of the elements.

Sensitive Data Masking

The Sensitive Data Masking feature allows message fields to be designated as holding sensitive data by adding the attribute *sensitive="1"* to the element describing the field. Fields so designated are masked in the diagnostic and HTTP logs and in memory. For example, a request or response containing the following dictionary structure:

```
<type name="CardHolderData">
    <element name="name" type="string" offset="0" size="32"/>
    <element name="cardNumber" type="string" size="32" sensitive="1"/>
</type>
```

Would appear in the diagnostic log as shown in the following fragments:

```
{
    "CardHolderData": {
        "name" : "John Doe",
        "cardNumber" : "****************"
    }
}
```

```
0x00000000: 4a6f 686e 2044 6f65 6800 0000 0000 0000  00000000: John Doe........
0x00000010: 0000 0000 0000 0000 0000 0000 0000 0000  00000016: ................
0x00000020: 2a2a 2a2a 2a2a 2a2a 2a2a 2a2a 2a2a 2a2a  00000032: ****************
0x00000030: 2a2a 2a2a 2a2a 2a2a 2a2a 2a2a 2a2a 2a2a  00000048: ****************
```

Working with Optional Elements

LightWave Server has several mechanisms to indicate the presence of optional elements. These mechanisms use a schema attribute to indicate the presence or absence of the element.

- Using the hideIfEmpty Attribute
- Using the isSet Attribute
- Using the minOccurs Attribute

Using the hideIfEmpty Attribute

The *hideIfEmpty="1"* attribute may be set on any element to indicate that if the source element is *empty*, it should not be included in the response payload. An element is considered empty when it is completely filled with the element's string padding character. Note that the string padding may be explicitly set on the element using the stringPadding attribute, or inherited from its parent. In the following example, if element *optionalIfSpaces* is filled with spaces, or *optionalIfZeros* is filled with zeros, the elements will not be included in the response payload.

```
<type name="SampleType">
    <element name="optionalIfSpaces" type="string" size="32" hideIfEmpty="1"
stringPadding="spaces" />
```

```
    <element name="optionalIfZeroes" type="aStructureType" hideIfEmpty="1"
stringPadding="zeros" />
</type>
```

Using the isSet Attribute

The *isSet* attribute may be set on any element, and references a target integer element which is used to indicate the presence or absence of the source element. In the following example, if the value of the element *isAddressSet* is non-zero, then the *theAddress* is included in the response. In this example, the *hide* attribute is set on the *isAddressSet* element to indicate that it is never included in the response payload. Note that when a response is received, the *isSet* target element will be set to zero if the source element is not present in the response, or non-zero if it is present.

```
<type name="SampleType">
    <element name="isAddressSet" type="int" size="4" hide="1" />
    <element name="theAddress" type="AddressType" isSet="isAddressSet" />
</type>
```

Using the minOccurs Attribute

> ℹ Using the minOccurs attribute has a mechanism to hide string elements is deprecated. New definitions should use *hideIfEmpty*.

The *minOccurs="0"* attribute may be set on elements with *type="string"* to indicate that if the source element is filled with the string padding character, it should not be included in the response payload. In the following example, if element *optionalIfSpaces* is filled with spaces, or *optionalIfZeros* is filled with zeros, the elements will not be included in the response payload.

```
<type name="SampleType">
    <element name="optionalIfSpaces" type="string" size="32" minOccurs="0"
stringPadding="spaces" />
    <element name="optionalIfZeros" type="string" size="32" minOccurs="0"
stringPadding="zeros" />
</type>
```

Using MEASURE Counters

LightWave Server supports several 'user defined' MEASURE counters which you can use to help analyze performance of your LightWave Server application. The available MEASURE counter names and types are shown below:

Counter Name	Type	Description
lw-connect	qbusy	The amount of time spent processing an inbound connection.
lw-conn-ssl-hs	qbusy	The amount of time spent processing the SSL/TLS handshake for a connection.
lw-request	qbusy	The amount of time spent receiving the request from the client over the network.
lw-deserialize	qbusy	The amount of time spent deserializing the request payload into the server request IPM.
lw-pathsend-io	qbusy	The amount of time spent sending the request and receiving the reply from a Pathway server.
lw-process-io	qbusy	The amount of time spent sending the request and receiving the reply from a process server.
lw-serialize	qbusy	The amount of time spent serializing the reply IPM into the response payload.
lw-request	qbusy	The amount of time spent sending the response to the client over the network
lw-diag-log	qbusy	The amount of time spent generating the diagnostic log.
lw-msg-log	qbusy	The amount of time spent filtering and delivering the message logging events.

Counter Name	Type	Description
lw-requests	accum	The total number of requests processed
lw-ssl-hs	accum	The total number of SSL/TLS handshakes
lw-recv-bytes	accum	The total number of bytes received over the network
lw-send-bytes	accum	The total number of bytes received over the network
lw-sock-recvs	accum	The total number of TCP/IP socket recv operations processed
lw-sock-sends	accum	The total number of TCP/IP socket send operations processed
lw-sock-connects	accum	The total number of TCP/IP connection requests processed.
lw-sock-accepts	accum	The total number of TCP/IP accept operations processed.
lw-deleg-fails	accum	The total number of SERVER to SWORKER requests delegations that have failed. This may indicate that the SWORKERs are over capacity.
lw-pathsend-ios	accum	The total number of Pathsend I/Os processed.
lw-process-ios	accum	The total number of process server I/Os processed.
lw-pathsend-max	snapshot	The maximum number of concurrent Pathsend I/O operations
lw-process-max	snapshot	The maximum number of concurrent process server I/O operations.

```
> VOLUME <lwc-installation-subvolume>
> MEASCOM

+ add processh server

+ add userdef server
+ add counter lw-connect        , process server, qbusy
+ add counter lw-conn-ssl-hs    , process server, qbusy
+ add counter lw-request        , process server, qbusy
+ add counter lw-deserialize    , process server, qbusy
+ add counter lw-pathsend-io    , process server, qbusy
+ add counter lw-process-io     , process server, qbusy
+ add counter lw-serialize      , process server, qbusy
+ add counter lw-response       , process server, qbusy
+ add counter lw-diag-log       , process server, qbusy
+ add counter lw-msg-log        , process server, qbusy
+ add counter lw-requests       , process server, accum
+ add counter lw-ssl-hs         , process server, accum
+ add counter lw-recv-bytes     , process server, accum
+ add counter lw-send-bytes     , process server, accum
+ add counter lw-sock-recvs     , process server, accum
+ add counter lw-sock-sends     , process server, accum
+ add counter lw-sock-connects  , process server, accum
+ add counter lw-sock-accepts   , process server, accum
+ add counter lw-deleg-fails    , process server, accum
+ add counter lw-pathsend-ios   , process server, accum
+ add counter lw-process-ios    , process server, accum
+ add counter lw-pathsend-max   , process server, snapshot
+ add counter lw-process-max    , process server, snapshot

+ start measurement LWSMEAS

== run your application

+ stop measurement LWSMEAS

+ list userdef *, rate off
```

These MEASURE counters may differ from values recorded in diagnostic logs. The diagnostic log measures elapsed time whereas the MEASURE counters represent actual work time.

Message Logging

The LightWave Server™ processes (SERVER & SWORKER) can be configured to collect and forward web service call data and statistics to a customer provided collector process. The information delivered to the collector process may include:

- Request / response inter-process messages (IPM).
- HTTP protocol information
- HTTP request / response bodies
- User data
- Request metadata

The collector process may use this information for any purpose, including but not limited to:

- Implement custom request logging
- Accumulate and report processing statistics
- Store message traffic for compliance purposes

Selected information for each request is sent to a specified collector target process as an inter-process message. Configuration details for the collector process, and request selection criteria in a Message Logging Configuration File.

> ℹ Collector processes must be Pathway serverclasses. Standalone processes are not supported.

The following sections describe how to configure and use Message Logging.

- The Message Log Request IPM
- Request Field Definitions
    - Header Fields
    - Message Data Map Field Types
    - Message Data Map Fields
- Log Request Streaming
- Logging User Data
- Logging Request Metadata
- Sensitive Data Masking
- Configuring Message Logging
- Checking Configuration File Syntax
- Configuration Reference
    - Collector Elements

> Hereinafter, the *sending process* refers to either the SERVER or SWORKER process, depending on which process is handling the client request.

The Message Log Request IPM

The sending process sends request information to the collector process using structures defined in the LWMLDDL DDL file. The request IPM consists of a standard header, a message data map containing offset and length information for the elements in the message data buffer, and a buffer containing the message data itself. The collector process decomposes the message data buffer using the information in the map. In the event that the Log Request IPM exceeds the maximum inter-process I/O size for the collector process, the request is streamed to the collector in multiple sequential requests.

Request Field Definitions

Header Fields

Field Name	Description
RQ-CODE	The standard request code for the Message Log request. This field will have the value LW-ML-RQ-REQUEST

Field Name	Description
RQ-VERSION	The Message Log request version.
RQ-LENGTH	The length in bytes of the entire Message Log request.
RQ-TS-UNIQUE	A 128-bit timestamp returned from the TS_UNIQUE_CREATE_ system procedure call that is unique to the system it is generated on and any system in the same EXPAND network

Message Data Map Field Types

The message map contains a list of fields describing data in the message data area. A map field may be one of the following types:

Data Type	Description
Buffer	This data type consists of a fixed length buffer containing binary data. Map fields for this data type include <field-name>-OFFSET and <field-name>-LEN. The -OFFSET element indicates the buffer offset from the start of the request IPM and the -LEN element indicates the length of the buffer in bytes. Fixed length buffer fields are guaranteed to be 4 byte aligned with the start of the IPM.
String	This data type consists of a NULL terminated character string. Map fields for this data type include <field-name>-OFFSET and <field-name>-LEN. The -OFFSET element indicates the offset of the string from the start of the request IPM and the -LEN element indicates the length of the string *including the NULL terminator*. String fields are byte aligned with the start of the IPM.
Collection	This data type consists of a collection of name / value pairs consisting of String data. Map fields for this data type include <field-name>-OFFSET, <field-name>-LEN, and <field-name>-COUNT. The -OFFSET element indicates the offset of the name / value collection from the start of the request IPM and the -LEN element indicates the length of the collection data *including all NULL terminators*. The -COUNT element indicates the number of name / value pairs in the collection.
Integer	A 32 or 64 bit integer value.

Data Type	Description
Timestamp	A 64 bit JULIANTIMESTAMP
Interval	A 32 bit unsigned integer containing a time interval in microseconds.

Message Data Map Fields

Field Name	Data Type	Description
START-TIME	Timestamp	The request start time.
END-TIME	Timestamp	The request end time.
TOTAL-TIME	Interval	Total request time in microseconds
CONNECT-TIME	Interval	Always 0.
CONNECT-HS-TIME	Interval	The TLS handshake time.
REQUEST-TIME	Interval	The request send time.
RESPONSE-TIME	Interval	The response time which consists of the service processing time and the response network transfer time.
SERIALIZE-TIME	Interval	Request serialization time.
DESERIALIZE-TIME	Interval	The response deserialization time.

Field Name	Data Type	Description
SERVER-IO-TIME	Interval	The process or serverclass I/O time.
RQ-IPM	Buffer	The application request IPM.
RQ-REQUEST-LINE	String	The HTTP request line.
RQ-METHOD	String	The request method, e.g, "POST".
RQ-URI	String	The request URI.
RQ-PARAMS	Collection	The request query string parameters.
RQ-HTTP-VER	String	The HTTP protocol version, e.g. "1.1".
RQ-HEADERS	Collection	The HTTP request headers.
RQ-BODY	Buffer	The HTTP request body
RP-IPM	Buffer	The application reply IPM
RP-STATUS	Integer	The HTTP response status.
RP-STATUS-LINE	String	The HTTP response line.
RP-HEADERS	Collection	The HTTP response headers.
RP-BODY	Buffer	The HTTP response body
USER-DATA	Collection	Any fields designated in the request IPM as User Data. See Logging User Data.

Field Name	Data Type	Description
METADATA	Collection	Any fields selected from the available request metadata. See Logging Request Metadata.

Log Request Streaming

The sending process writes the Message Log request buffer to the collector process using the largest inter-process message size available. This may be anywhere from 32K to 2MB, depending on the system configuration. In the event that the Message Log request is larger than the maximum size allowed, it will be sent to the collector in multiple consecutive requests within a Pathway Dialog. The collector process should issue a sufficient number of $RECEIVE read operations in order to read the entire request.

Logging User Data

Application request IPM data may be included in the Message Log request by indicating which fields to include in the API definition type schema. Fields with the *msgLogUserData* schema property set to *true* will be included in the USER-DATA collection. Note that:

- Only primitive data types (string, integer, etc.) can be logged. Object and array types cannot be logged.
- All data regardless of IPM type is converted to string for inclusion in the USER-DATA collection.

The following schema fragment illustrates how to include application request data in the log request.

```
<type name="RequestMessage>
  <element name="requestData" type="string" size="32" />
  <element name="requestId" type="string" size="32" msgLogUserData="1" />
</type>
```

Logging Request Metadata

Message Logging may be configured to include additional request metadata. The metadata includes information such as sending process information and API request information. The following metadata is available.

Name	Value
api-name	The API name.
api-file-name	The API file name.
api-method	The API operation method.
api-path	The API operation path.
api-base-url	The API base url.
api-alias	The API operation alias
config-*	An item for each of the configuration options for the collector, e.g, config-collector-name, config-collector-pathmon, etc. Also the name of the filter selected, as config-filter-name.
net-local-ip	The local IP address of the socket connection.
net-local-port	The local port of the socket connection.
net-remote-ip	The remote IP address of the socket connection.
net-remote-port	The remote port of the socket connection.
net-process	The TCP/IP process in use.
process-name	The process name of the sending process.
process-program-file	The sending process program file name.
process-ancestor-name	The process name of the sending process ancestor.

Name	Value
process-ancestor-program	The program file name of the sending process ancestor.
tls-version	The TLS protocol version.
tls-cipher	The TLS cipher

Sensitive Data Masking

The Sensitive Data Masking feature allows message fields to be designated as holding sensitive data by adding the attribute "*sensitive" : true* to the element describing the field. Fields so designated are masked in the Message Log request. The feature can be disable by starting the SERVER process with the *--disable-sensitive-data-masking* option. For more information see Sensitive Data Masking.

Configuring Message Logging

Message Logging is configured using a process configuration file. The configuration is activated by specifying the configuration options in an EDIT file and providing the file location with the --msg-log startup option. Change monitoring for the configuration file may be configured using the --monitor option, for example:

```
tacl> run SERVER --msg-log +$vol.subvol.mlogcfg [ --monitor msg-log ]
```

The configuration is specified using YAML 1.1 format. Multiple collector processes and message selection rules may be configured, for example:

```
---
msg-log:
  collectors:
    collector-all-content:
      enabled: true
      type: serverclass
      pathmon: $lwml
      serverclass: collect-all
      content: all
    collector-http-only:
      enabled: true
      type: serverclass
      pathmon: $lwml
      serverclass: collect-http
      content: rq-http,rp-http
  filters:
    successful-request:
      # Log HTTP info for successful requests
      enabled: true
      match-all:
        http-status: 200:299
        reply-code: 0
      collectors: collector-http-only
    failed-request:
      # Log all info for failed requests
      enabled: true
      match-any:
        http-status: 300:600
        reply-code: 1,1
      collectors: collector-all-content
    catch-all:
      # Illustrate catch all rule that sends to both collectors.
      enabled: true
      match-always: true
      collectors:
        - collector-http-only
        - collector-all-content
...
```

More information on YAML can be found on the official YAML web site and Wikipedia.

Checking Configuration File Syntax

During the process of creating or updating a configuration file, the configuration may be validated using the LWSCOM CHECK command.

```
tacl> run lwscom
LightWave Server COM 1.1.0 - \NODE.$X123
Copyright (c) 2020 NuWave Technologies, Inc. All rights reserved.
LWSCOM 1-> check config mlogcfg
Checking CONFIG file MLOGCFG.
The file contains a valid 'msg-log' configuration.
The CONFIG file is valid.
LWSCOM 2->
```

Configuration Reference

The basic structure of the configuration file is

```
---
```

```
# A YAML document begins with '---' and end with '...'
# The msg-log configuration begins with a 'msg-log' element.
msg-log:
  collectors:
    # The 'collectors' element contains a collection of named collector definitions.
    # A collector name is 1-80 characters, begins with a-z, may contain a-z, 0-9, '-', '_', '.', must end with a-z or 0-9, and must be unique among collectors.
    collector-name:
      # Collector elements
  filters:
    # The 'filters' element contains a collection of named filter definitions. Filters are checked in order until a match is found.
    # A filter name is 1-80 characters, begins with a-z, may contain a-z, 0-9, '-', '_', '.', must end with a-z or 0-9, and must be unique among filters.
    filter-name: # Collector name is 1-80 characters, begins with a-z, may contain a-z, 0-9, '-', '_', '.', must end with a-z or 0-9.
      # Filter elements
...
```

Collector Elements

Option	Description
enabled	A boolean such as "true" or "false", indicating if the collector is enabled. Disabled collectors may be referenced by filters but messages will not be sent. Required.
type	The type of collector, which must be "serverclass". If omitted, the default value is "serverclass".
pathmon	The process name of the serverclass PATHMON process, which may be specified as a DEFINE. Required if type is "serverclass".
serverclass	The serverclass name. Required if type is "serverclass".

Option	Description
content	A comma separated list of one or more of the following content selectors: • api - API metadata • net - Network metadata • process - Process metadata • rq-ipm - The request IPM • rq-http - The request HTTP protocol information (request line, headers, etc.) • rq-body - The request body • rp-ipm - The reply IPM • rp-http - The reply HTTP protocol information (request line, headers, etc). • rp-body- The reply body • tls - TLS metadata • user - The user data. • all - all possible content. The default value is 'all'. May be negated, see Using Negation.
io-retry-interval	The number of seconds to wait before retrying a failed I/O to the collector target. The value must be in the range 5 - 30 seconds.
io-retry-limit	The number of times to retry a failed I/O to the collector target. The value must be in the range 1 - 10 retries.
io-max-length	The maximum size of any I/O to the collector target. The sending process always tries to use the maximum I/O size possible for the target collector. This parameter may be used to reduce the I/O size to any limit imposed by the target process. Note that this parameter may also be used to artificially limit the I/O size of a Pathsend serverclass collector, in order to test the servers ability to handle streaming requests using Pathway dialogs.
request-code	The IPM request code to use for the message logging request. By default, the the value of LW-ML-RQ-MSG-LOG as the IPM request code. This option may be used to override that value and allow each collector configuration to use a unique request code.

Filter Elements

Option	Description
enabled	A boolean such as "true" or "false", indicating if the filter is enabled. Disabled filters are ignored.
continue-on-match	A boolean such as "true" or "false", indicating if filter processing should continue if the current filter matches. By default, when a filter matches, no further filters are checked. If omitted, the default value is "false".
match-all	The match-all element contains a collection of filter rules. In order for the filter to match, all rules must match. Optional.
match-any	The match-any element contains a collection of filter rules. In order for the filter to match, any of the rules must match. Optional.
match-always	A boolean such as "true" or "false", indicating that the filter always matches. Optional. If present, the filter matches and any match-all or match-any collections are ignored. This option can be used to create a "catch all" filter.
collectors	A single collector name or a sequence of collector names. If the filter rules match, a logging request will be sent to each collector.

Note that a filter may contain both the *match-all* and *match-any* options. When both are present, they must both match in order for the filter to match (they are logically ANDed).

Filter Rule Elements

Enclosing rule element values in double quotes is strongly recommended.

In the following examples, one or more required filter elements may have been omitted for brevity.

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>alias</td><td>

A single pattern or a sequence of patterns used to select or ignore an operation alias. See Using Patterns. May be negated, see Using Negation.

Example

```
filters:
  match-when-alias-starts-with-put:
    match-all:
      alias: "put*"
  match-when-alias-starts-with-post-or-get:
    match-all:
      alias:
        - "post*"
        - "get*"
```

</td></tr>
<tr><td>http-status</td><td>

A single range or a sequence of ranges used to select or ignore http-status values. See Using Patterns. May be negated, see Using Negation.

Example

```
filters:
  match-when-in-200-range:
    match-all:
      http-status: "200:299"
  match-when-in-200-range-but-not-222:
    match-all:
      http-status:
        - "200:299"
        - "!222"
```

</td></tr>
</table>

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>method</td><td>

A comma separated list or pattern used to select or ignore HTTP methods. See Using Patterns. May be negated, see Using Negation.

Example

```
filters:
  match-when-post-or-put:
    match-all:
      method: "post,put"
  match-when-not-get:
    match-all:
      method: "!get"
```

</td></tr>
<tr><td>path</td><td>

A single pattern or sequence of patterns used to select or ignore an operation path. See Using Patterns. May be negated, see Using Negation. Note that the path used for matching is the actual path used to invoke the operation, not the path specified in the API definition.

Example

```
filters:
  match-when-account-request:
    match-all:
      path: "/api/1/account/*"
  match-when-account-or-customer-request:
    match-all:
      path:
        - "/api/v1/account/*"
        - "/api/v1/customer/*"
```

</td></tr>
</table>

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>reply-code</td><td>
<p>A single range or sequence of ranges used to select or ignore reply-code values. See Using Patterns. May be negated, see Using Negation.</p>
<p><b>Example</b></p>
<pre><code>filters:
  match-when-0:
    match-all:
      reply-code: "0"
  match-when-10-20-but-not-15:
    match-all:
      reply-code:
        - "!15"
        - "10:20"
</code></pre>
</td></tr>
<tr><td>rq-header / rp-header</td><td>
<p>A collection of HTTP request or response header name value pairs. The name matches the HTTP header name. The value may be a pattern. See Using Patterns. May be negated, see Using Negation.</p>
<p>Boolean true of false may also be specified to match when the header is or is not present.</p>
<p><b>Example</b></p>
<pre><code>filters:
  match-when-json-reply:
    match-all:
      rp-header:
        content-type: application/json
        header-1: false   # match if header-1 is not present
        header-2: true   # match if header-2 is present, regardless
of value
</code></pre>
</td></tr>
</table>

Using Patterns

A pattern may be a literal string, a simple pattern, or a Perl compatible regular expression.

- literal string - A string that exactly matches the element value. The string comparison is case insensitive.

- simple pattern - A string that may contain the question mark ('?') character to match any single character or the asterisk ('*') to match multiple characters. The string comparison is case insensitive.
- regular expression - A string prefixed with "regex:" which contains a Perl compatible regular expression used to match the element value. See Perl Compatible Regular Expressions.

For example, the following rules all match the path "/acme/account/9001"

```
path: "/acme/account/9001"  # literal pattern
path: "/acme/account/*"     # match any account
path: "regex:(?i)^\/acme\/account\/(9001|9002)$" # match account 9001 or 9002
```

Using Ranges

A range of values is specified as a comma separated list of single integers, or two integers separated by a colon ':' indicating an inclusive range. For example, the range "1,2,5:10" matches the values 1, 2, and 5-10 inclusive.

Using Negation

Where noted, patterns and ranges may be negated in order to indicate that the rule should match when the pattern or range does NOT match. Negation is indicated by prefixing the pattern or range with the exclamation mark ('!'). Note that because the exclamation mark is a special character in YAML, patterns or ranges specified with negation must be enclosed in double quotes. For example:

```
path: "!/acme/account/9001"  # match when this is NOT the path.
http-status: !200:299        # match when the HTTP status is NOT in the 200-299 range
rp-header:
  content-type: "!application/json"   # match when the content-type reply header is
NOT "application/json"
```

Sample Collectors

Sample collectors processes in C and COBOL can be found on Github.

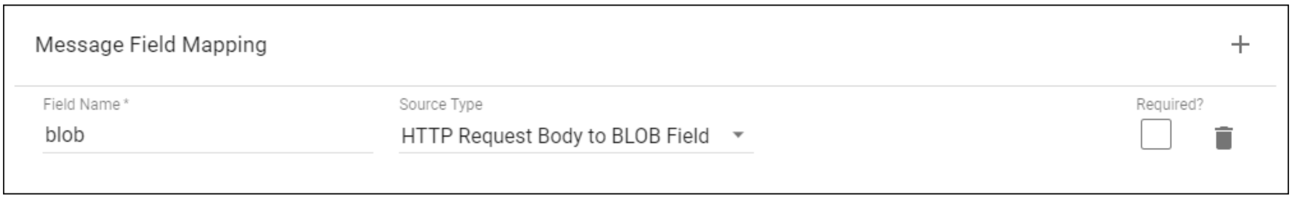
Using BLOBs

There may be cases where an application requires more control over the Web service request or response payload. For example, the Web service requires a message structure that can't be created or parsed by the native LightWave Server JSON support, or the message payload is larger than the maximum message size allowed by an interprocess message. For these cases LightWave Server supports the BLOB data type.

When using BLOBs (Binary Large Objects), your application provides or receives the Web service payload as raw data in the IPM or in an Enscribe file. When a file is used, the name of the file is supplied in the request and/or response IPMs.

Using IPM BLOBs

When an API definition specifies a request or reply field type as *BLOB Field*, the data transferred to or from the field in the IPM represents exactly the data that is to be sent or that was received. The *sizeIs* attribute may be used to reference a field that contains the actual size of the data to send or the size of the data that was received. The maximum size of an IPM BLOB is 2MB, minus the size of all other fields in the IPM.



Message Field Mapping

Field Name *: blob

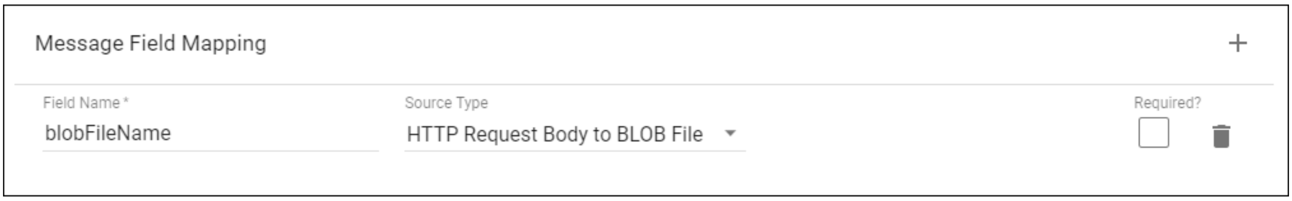
Source Type: HTTP Request Body to BLOB Field

Required? ☐

```
<type name="RqBlob">
  <element name="rqCode" type="short" offset="0" size="2"/>
  <element name="blobSize" type="short" offset="2" size="2" hide="1"/>
  <element name="blob" type="string" offset="4" size="1000000" sizeIs="blobSize"/>
</type>
```

Using File BLOBs

File BLOBs are similar to IPM BLOBs but Enscribe files are used to transfer the data. When an API definition specifies the request or reply field type as *BLOB File*, the source or destination field contains the name of the Enscribe file containing the BLOB data. For web service requests, LightWave Server creates the BLOB file and supplies the file name in the blobFileName field in the request IPM to the application server. For web service responses, the application server supplies the BLOB file name in the reply IPM. The LightWave Server SERVER / SWORKER processes must have read access to the file.



Message Field Mapping

Field Name *: blobFileName

Source Type: HTTP Request Body to BLOB File

Required? ☐

```
<type name="RqBlobFile">
  <element name="rqCode" type="short" offset="0" size="2"/>
  <element name="blobFileName" type="string" offset="2" size="256"/>
</type>
```

Request BLOB files are created by the LightWave Server processes as type 101 EDIT files, when the request content type is "text/plain". For all other content types, type 180 files (binary stream) are created. The files are created using the location and attributes specified in the SERVER process *--blob-files* startup option. See SERVER for more information on the *--blob-files* startup option. The location and attributes may also be specified in the API definition. Values specified in the API definition are optional, and when specified will override the SERVER process values.

> Specifying the BLOB file location and attributes in the API definition can be problematic if the API definition will be moved to different environments on different NonStop systems. When specifiying the attributes in the API definition, consider using DEFINEs with unique application specific names to specify the BLOB file location. The DEFINE values can be set appropriately in each environment when starting the SERVER process. Use SAFEGUARD to configure appropriate subvolume access permissions to allow access to the BLOB files by users and applications.

Note that the application is responsible for the disposal of request BLOB files. Response BLOB files may be automatically purged by LightWave Server by selecting the *Purge BLOB file on request completion* option in the API reply definition. When this option is selected, the BLOB file is deleted on completion of the request, regardless of the outcome.

Working with Timestamps

LightWave Server supports 64-bit and 48-bit timestamp fields. To define a timestamp field in schema, set the field type to "timestamp" and use the size attribute to indicate if the field is 64 bit or 48 bit, for example:

```
<type name="TimestampTypes">
  <element name="timestamp64bitJulian" type="timestamp" size="8" />
  <element name="timestamp48bit" type="timestamp" size="6" />
</type>
```

Timestamp Formats

Several standard timestamp formats can be serialized or deserialized from the message body. The timestamp format is specified using the *timestampFormat* attribute. The following formats are supported:

timestampFormat	description
ISO8601	ISO8601 Date/Time format as described in RFC 3339. This is the default format used when the timestampFormat property is omitted.
ISO8601:full-date	ISO8601 Date/Time full-date format as described in RFC 3339.
RFC2822	RFC2822 Date and Time format as desribed in RFC 2822
XSD:dateTime	XSD dateTime format as described in XML Schema Part 2: Datatypes

Schema definition with ISO8601:full-date

```
<type name="ISO8601Timestamp">
  <element name="theDate" type="timestamp" size="8" timestampFormat="ISO8601:full-date" />
</type>
```

Troubleshooting

There are a number of client-side tools such as soapUI and Fiddler that can be used to invoke client application requests and view the request and response messages. When developing user-defined APIs, you can also use the built in method tester which is available in the API editor. However, none of these tools provide insight into what is happening within LightWave Server. Use the following tools to troubleshoot server-side operation of your services.

- Diagnostic Logging

Diagnostic Logging

Diagnostic logging shows detailed information about the HTTP request and responses and the resulting inter-process messages and subsystem I/O processing related to an API request.

Diagnostic logging is resource intensive and will degrade the performance of all services on your LightWave Server instance. Diagnostic logging should not be enabled on production instances unless necessary. Use of the *Auto-disable timer* option is strongly recommended to ensure that Diagnostic logging is not left enabled indefinitely.

To enable diagnostic logging for a service

1. Turn on diagnostic logging in the Diagnostic Logs view. See Diagnostic Logs.

Character Encoding Names

LightWave Server uses IBM International Components for Unicode (ICU) to provide string translation services when deserializing and serializing Web service requests and responses. ICU allows the character encoding name to be specified using one of the various standard encoding names in addition to a number of aliases. More information on the ICU project can be found at http://site.icu-project.org.

LightWave Server supports the following character encoding names. Any of these names may be provided as the value of the *encoding* attribute.

```
Adobe-Standard-Encoding
ANSI_X3.4-1968
ANSI_X3.4-1986
ANSI1251
arabic
ASCII
ascii7
ASMO-708
Big5
big5hk
Big5-HKSCS
big5-hkscs:unicode3.0
BOCU-1
CCSID00858
CCSID01140
CCSID01141
CCSID01142
CCSID01143
CCSID01144
CCSID01145
CCSID01146
CCSID01147
```

```
CCSID01148
CCSID01149
CESU-8
chinese
cns11643
COMPOUND_TEXT
CP00858
CP01140
CP01141
CP01142
CP01143
CP01144
CP01145
CP01146
CP01147
CP01148
CP01149
cp037
cp1006
cp1025
cp1026
cp1047
cp1089
cp1097
cp1098
cp1112
cp1122
cp1123
cp1124
cp1125
cp1131
cp1140
cp1141
cp1142
cp1143
cp1144
cp1145
cp1146
cp1147
cp1148
cp1149
cp1200
cp1201
cp1208
cp1250
cp1251
cp1252
cp1253
```

```
cp1254
cp1255
cp1256
cp1257
cp1258
cp1363
cp1383
cp1386
cp273
cp277
cp278
cp280
cp284
cp285
cp290
cp297
cp33722
cp367
cp37
cp420
cp424
cp437
cp500
cp737
cp775
cp803
cp813
cp819
cp838
cp850
cp851
cp852
cp855
cp856
cp857
cp858
cp860
cp861
cp862
cp863
cp864
cp865
cp866
CP868
cp869
CP870
CP871
cp874
```

```
cp875
cp878
cp912
cp913
cp914
cp915
cp916
cp918
cp920
cp921
cp922
cp923
cp930
cp932
cp933
cp935
cp936
cp937
cp939
cp943
cp943c
cp949
cp949c
cp950
cp964
cp970
cp-ar
cp-gr
cpibm284
cpibm285
cpibm297
cpibm37
cp-is
csAdobeStandardEncoding
csASCII
csBig5
csBOCU-1
csEUCKR
csEUCPkdFmtJapanese
csGB2312
csHPRoman8
csIBM037
csIBM1026
csIBM273
csIBM277
csIBM278
csIBM280
csIBM284
```

```
csIBM285
csIBM290
csIBM297
csIBM420
csIBM424
csIBM500
csIBM855
csIBM857
csIBM860
csIBM861
csIBM863
csIBM864
csIBM865
csIBM866
csIBM868
csIBM869
csIBM870
csIBM871
csIBM918
csIBMThai
csISO2022CN
csISO2022JP
csISO2022JP2
csISO2022KR
csISO58GB231280
csISOLatin0
csISOLatin1
csISOLatin2
csISOLatin3
csISOLatin4
csISOLatin5
csISOLatin6
csISOLatin9
csISOLatinArabic
csISOLatinCyrillic
csISOLatinGreek
csISOLatinHebrew
csJISEncoding
csKOI8R
csKSC56011987
csMacintosh
csPC775Baltic
csPC850Multilingual
csPC851
csPC862LatinHebrew
csPC8CodePage437
csPCp852
csPCp855
```

```
csShiftJIS
csUCS4
csUnicode
csWindows31J
cyrillic
DOS-720
DOS-862
ebcdic-ar
ebcdic-cp-ar1
ebcdic-cp-ar2
ebcdic-cp-be
ebcdic-cp-ca
ebcdic-cp-ch
EBCDIC-CP-DK
ebcdic-cp-es
ebcdic-cp-fi
ebcdic-cp-fr
ebcdic-cp-gb
ebcdic-cp-he
ebcdic-cp-is
ebcdic-cp-it
ebcdic-cp-nl
EBCDIC-CP-NO
ebcdic-cp-roece
ebcdic-cp-se
ebcdic-cp-us
ebcdic-cp-wt
ebcdic-cp-yu
ebcdic-de
ebcdic-de-273+euro
ebcdic-dk
ebcdic-dk-277+euro
ebcdic-es-284+euro
ebcdic-fi-278+euro
ebcdic-fr-297+euro
ebcdic-gb
ebcdic-gb-285+euro
ebcdic-he
ebcdic-international-500+euro
ebcdic-is
ebcdic-is-871+euro
ebcdic-it-280+euro
EBCDIC-JP-kana
ebcdic-no-277+euro
ebcdic-se-278+euro
ebcdic-sv
ebcdic-us-37+euro
ebcdic-xml-us
```

```
ECMA-114
ECMA-118
ECMA-128
ELOT_928
EUC-CN
eucjis
EUC-JP
euc-jp-2007
EUC-KR
eucTH
EUC-TW
euc-tw-2014
Extended
GB_2312-80
gb18030
GB18030
GB2312
GB2312.1980-0
gb2312-1980
GBK
greek
greek8
hebrew
hebrew8
hkbig5
HKSCS-BIG5
hp15CN
hp-roman8
HZ
HZ-GB-2312
IBM00858
IBM01140
IBM01141
IBM01142
IBM01143
IBM01144
IBM01145
IBM01146
IBM01147
IBM01148
IBM01149
IBM037
ibm-037
IBM1006
ibm-1006
ibm-1006_P100-1995
ibm-1025
ibm-1025_P100-1995
```

```
IBM1026
ibm-1026
ibm-1026_P100-1995
IBM1047
ibm-1047
IBM1047_LF
ibm-1047_P100-1995
ibm-1047_P100-1995,swaplfnl
ibm-1047-s390
ibm-1051
ibm-1051_P100-1995
ibm-1089
ibm-1089_P100-1995
ibm-1097
ibm-1097_P100-1995
IBM1098
ibm-1098
ibm-1098_P100-1995
ibm-1112
ibm-1112_P100-1995
ibm-1122
ibm-1122_P100-1999
ibm-1123
ibm-1123_P100-1995
ibm-1124
ibm-1124_P100-1996
ibm-1125
ibm-1125_P100-1997
ibm-1129
ibm-1129_P100-1997
ibm-1130
ibm-1130_P100-1997
ibm-1131
ibm-1131_P100-1997
ibm-1132
ibm-1132_P100-1998
ibm-1133
ibm-1133_P100-1997
ibm-1137
ibm-1137_P100-1999
ibm-1140
ibm-1140_P100-1997
ibm-1140_P100-1997,swaplfnl
ibm-1140-s390
ibm-1141
IBM1141_LF
ibm-1141_P100-1997
ibm-1141_P100-1997,swaplfnl
```

```
ibm-1141-s390
ibm-1142
ibm-1142_P100-1997
ibm-1142_P100-1997,swaplfnl
ibm-1142-s390
ibm-1143
ibm-1143_P100-1997
ibm-1143_P100-1997,swaplfnl
ibm-1143-s390
ibm-1144
ibm-1144_P100-1997
ibm-1144_P100-1997,swaplfnl
ibm-1144-s390
ibm-1145
ibm-1145_P100-1997
ibm-1145_P100-1997,swaplfnl
ibm-1145-s390
ibm-1146
ibm-1146_P100-1997
ibm-1146_P100-1997,swaplfnl
ibm-1146-s390
ibm-1147
ibm-1147_P100-1997
ibm-1147_P100-1997,swaplfnl
ibm-1147-s390
ibm-1148
ibm-1148_P100-1997
ibm-1148_P100-1997,swaplfnl
ibm-1148-s390
ibm-1149
ibm-1149_P100-1997
ibm-1149_P100-1997,swaplfnl
ibm-1149-s390
IBM1153
ibm-1153
ibm-1153_P100-1999
ibm-1153_P100-1999,swaplfnl
ibm-1153-s390
ibm-1154
ibm-1154_P100-1999
ibm-1155
ibm-1155_P100-1999
ibm-1156
ibm-1156_P100-1999
ibm-1157
ibm-1157_P100-1999
ibm-1158
ibm-1158_P100-1999
```

```
ibm-1160
ibm-1160_P100-1999
ibm-1162
ibm-1162_P100-1999
ibm-1164
ibm-1164_P100-1999
ibm-1168
ibm-1168_P100-2002
ibm-1200
ibm-1201
ibm-1202
ibm-1203
ibm-1204
ibm-1205
ibm-1208
ibm-1209
ibm-1212
ibm-1213
ibm-1214
ibm-1215
ibm-1232
ibm-1233
ibm-1234
ibm-1235
ibm-1236
ibm-1237
ibm-1250
ibm-1250_P100-1995
ibm-1251
ibm-1251_P100-1995
ibm-1252
ibm-1252_P100-2000
ibm-1253
ibm-1253_P100-1995
ibm-1254
ibm-1254_P100-1995
ibm-1255
ibm-1255_P100-1995
ibm-1256
ibm-1256_P110-1997
ibm-1257
ibm-1257_P100-1995
ibm-1258
ibm-1258_P100-1997
ibm-12712
ibm-12712_P100-1998
ibm-12712_P100-1998,swaplfnl
ibm-12712-s390
```

```
ibm-1276
ibm-1276_P100-1995
ibm-13488
ibm-13489
ibm-13490
ibm-13491
ibm-13496
ibm-13497
ibm-1363
ibm-1363_P110-1997
ibm-1363_P11B-1998
ibm-1363_VASCII_VSUB_VPUA
ibm-1363_VSUB_VPUA
ibm-1364
ibm-1364_P110-2007
ibm-1371
ibm-1371_P100-1999
ibm-1373
ibm-1373_P100-2002
ibm-1375
ibm-1375_P100-2008
ibm-1383
ibm-1383_P110-1999
ibm-1383_VPUA
ibm-1386
ibm-1386_P100-2001
ibm-1386_VSUB_VPUA
ibm-1388
ibm-1388_P103-2001
ibm-1390
ibm-1390_P110-2003
ibm-1392
ibm-1399
ibm-1399_P110-2003
ibm-16684
ibm-16684_P110-2003
ibm-16804
ibm-16804_X110-1999
ibm-16804_X110-1999,swaplfnl
ibm-16804-s390
ibm-17584
ibm-17585
ibm-17586
ibm-17587
ibm-17592
ibm-17593
ibm-20780
ibm-21680
```

```
ibm-21681
ibm-21682
ibm-21683
ibm-25546
ibm-25776
ibm-25777
ibm-25778
ibm-25779
IBM273
ibm-273
ibm-273_P100-1995
IBM277
ibm-277
ibm-277_P100-1995
IBM278
ibm-278
ibm-278_P100-1995
IBM280
ibm-280
ibm-280_P100-1995
IBM284
ibm-284
ibm-284_P100-1995
IBM285
ibm-285
ibm-285_P100-1995
IBM290
ibm-290
ibm-290_P100-1995
IBM297
ibm-297
ibm-297_P100-1995
ibm-29872
ibm-29873
ibm-29874
ibm-29875
ibm-33722
ibm-33722_P120-1999
ibm-33722_P12A_P12A-2009_U2
ibm-33722_VASCII_VPUA
ibm-33722_VPUA
IBM367
ibm-367
ibm-37
ibm-37_P100-1995
ibm-37_P100-1995,swaplfnl
ibm-37-s390
IBM420
```

```
ibm-420
ibm-420_X120-1999
IBM424
ibm-424
ibm-424_P100-1995
IBM437
ibm-437
ibm-437_P100-1995
ibm-4517
ibm-4517_P100-2005
ibm-4899
ibm-4899_P100-1998
ibm-4902
ibm-4909
ibm-4909_P100-1999
ibm-4971
ibm-4971_P100-1999
IBM500
ibm-500
ibm-500_P100-1995
ibm-5012
ibm-5012_P100-1999
ibm-5026
ibm-5035
ibm-5050
ibm-5054
ibm-5123
ibm-5123_P100-1999
ibm-5304
ibm-5305
ibm-5346
ibm-5346_P100-1998
ibm-5347
ibm-5347_P100-1998
ibm-5348
ibm-5348_P100-1997
ibm-5349
ibm-5349_P100-1998
ibm-5350
ibm-5350_P100-1998
ibm-5351
ibm-5351_P100-1998
ibm-5352
ibm-5352_P100-1998
ibm-5353
ibm-5353_P100-1998
ibm-5354
ibm-5354_P100-1998
```

```
ibm-5471
ibm-5471_P100-2006
ibm-5478
ibm-5478_P100-1995
ibm-61955
ibm-61956
ibm-65025
ibm-720
ibm-720_P100-1997
IBM737
ibm-737
ibm-737_P100-1997
IBM775
ibm-775
ibm-775_P100-1996
ibm-803
ibm-803_P100-1999
ibm-813
ibm-813_P100-1995
IBM819
ibm-819
IBM838
ibm-838
ibm-838_P100-1995
ibm-8482
ibm-8482_P100-1999
IBM850
ibm-850
ibm-850_P100-1995
IBM851
ibm-851
ibm-851_P100-1995
IBM852
ibm-852
ibm-852_P100-1995
IBM855
ibm-855
ibm-855_P100-1995
IBM856
ibm-856
ibm-856_P100-1995
IBM857
ibm-857
ibm-857_P100-1995
ibm-858
ibm-858_P100-1997
IBM860
ibm-860
```

```
ibm-860_P100-1995
IBM861
ibm-861
ibm-861_P100-1995
IBM862
ibm-862
ibm-862_P100-1995
IBM863
ibm-863
ibm-863_P100-1995
IBM864
ibm-864
ibm-864_X110-1999
IBM865
ibm-865
ibm-865_P100-1995
IBM866
ibm-866
ibm-866_P100-1995
ibm-867
ibm-867_P100-1998
IBM868
ibm-868
ibm-868_P100-1995
IBM869
ibm-869
ibm-869_P100-1995
IBM870
ibm-870
ibm-870_P100-1995
IBM871
ibm-871
ibm-871_P100-1995
ibm-874
ibm-874_P100-1995
IBM875
ibm-875
ibm-875_P100-1995
ibm-878
ibm-878_P100-1996
ibm-9005
ibm-9005_X110-2007
ibm-901
ibm-901_P100-1999
ibm-902
ibm-902_P100-1999
ibm-9030
ibm-9066
```

```
ibm-9067
ibm-9067_X100-2005
ibm-912
ibm-912_P100-1995
ibm-913
ibm-913_P100-2000
ibm-914
ibm-914_P100-1995
ibm-915
ibm-915_P100-1995
ibm-916
ibm-916_P100-1995
IBM918
ibm-918
ibm-918_P100-1995
ibm-920
ibm-920_P100-1995
ibm-921
ibm-921_P100-1995
IBM922
ibm-922
ibm-922_P100-1999
ibm-923
ibm-923_P100-1998
IBM930
ibm-930
ibm-930_P120-1999
ibm-931
ibm-932
ibm-932_VSUB_VPUA
ibm-933
ibm-933_P110-1995
ibm-935
ibm-935_P110-1999
ibm-937
ibm-937_P110-1999
IBM939
ibm-939
ibm-939_P120-1999
ibm-9400
ibm-942
ibm-942_P12A-1999
ibm-942_VSUB_VPUA
ibm-9424
ibm-943
ibm-943_P130-1999
ibm-943_P15A-2003
ibm-943_VASCII_VSUB_VPUA
```

```
ibm-943_VSUB_VPUA
IBM-943C
ibm-9447
ibm-9447_P100-2002
ibm-9448
ibm-9448_X100-2005
ibm-9449
ibm-9449_P100-2002
ibm-949
ibm-949_P110-1999
ibm-949_P11A-1999
ibm-949_VASCII_VSUB_VPUA
ibm-949_VSUB_VPUA
IBM-949C
ibm-950
ibm-950_P110-1999
ibm-954
ibm-954_P101-2007
ibm-9580
ibm-964
ibm-964_P110-1999
ibm-964_VPUA
ibm-970
ibm-970_P110_P110-2006_U2
ibm-970_VPUA
ibm-971
ibm-971_P100-1995
ibm-971_VPUA
ibm-eucCN
IBM-eucJP
ibm-eucKR
ibm-eucTW
IBM-Thai
IMAP-mailbox-name
ISCII,version=0
ISCII,version=1
ISCII,version=2
ISCII,version=3
ISCII,version=4
ISCII,version=5
ISCII,version=6
ISCII,version=7
ISCII,version=8
iscii-bng
iscii-dev
iscii-guj
iscii-gur
iscii-knd
```

```
iscii-mlm
iscii-ori
iscii-tlg
iscii-tml
ISO_2022,locale=ja,version=0
ISO_2022,locale=ja,version=1
ISO_2022,locale=ja,version=2
ISO_2022,locale=ja,version=3
ISO_2022,locale=ja,version=4
ISO_2022,locale=ko,version=0
ISO_2022,locale=ko,version=1
ISO_2022,locale=zh,version=0
ISO_2022,locale=zh,version=1
ISO_2022,locale=zh,version=2
iso_646.irv:1983
ISO_646.irv:1991
ISO_8859-1:1987
ISO_8859-10:1992
ISO_8859-14:1998
ISO_8859-2:1987
ISO_8859-3:1988
ISO_8859-4:1988
ISO_8859-5:1988
ISO_8859-6:1987
ISO_8859-7:1987
ISO_8859-8:1988
ISO_8859-9:1989
ISO-10646-UCS-2
ISO-10646-UCS-4
ISO-2022-CN
ISO-2022-CN-CNS
ISO-2022-CN-EXT
ISO-2022-JP
ISO-2022-JP-1
ISO-2022-JP-2
ISO-2022-KR
ISO646-US
iso-8859_10-1998
iso-8859_11-2001
iso-8859_14-1998
iso8859_15_fdis
ISO-8859-1
ISO-8859-10
ISO-8859-11
ISO-8859-13
ISO-8859-14
ISO-8859-15
ISO-8859-2
```

```
ISO-8859-3
ISO-8859-4
ISO-8859-5
ISO-8859-6
ISO-8859-6-E
ISO-8859-6-I
ISO-8859-7
ISO-8859-8
ISO-8859-8-E
ISO-8859-8-I
ISO-8859-9
iso-celtic
iso-ir-100
iso-ir-101
iso-ir-109
iso-ir-110
iso-ir-126
iso-ir-127
iso-ir-138
iso-ir-144
iso-ir-148
iso-ir-149
iso-ir-157
iso-ir-199
iso-ir-58
iso-ir-6
JIS
JIS_Encoding
JIS7
JIS8
koi8
KOI8-R
KOI8-U
korean
KS_C_5601-1987
KS_C_5601-1989
ksc
KSC_5601
l1
l2
l3
l4
l5
l6
l8
l9
latin0
latin1
```

```
latin2
latin3
latin4
latin5
latin6
latin8
latin9
lmbcs
LMBCS-1
mac
macce
maccentraleurope
maccy
mac-cyrillic
macgr
macintosh
macos-0_2-10.2
macos-29-10.2
macos-35-10.2
macos-6_2-10.4
macos-7_3-10.2
macroman
mactr
MS_Kanji
MS874
ms932
MS936
ms949
ms950
MS950_HKSCS
pck
PC-Multilingual-850+euro
r8
roman8
SCSU
Shift_JIS
shift_jis78
sjis
sjis78
sun_eu_greek
thai8
TIS-620
tis620.2533
turkish
turkish8
ucs-2
ucs-4
ujis
```

```
unicode
unicode-1-1-utf-7
unicode-1-1-utf-8
unicode-2-0-utf-7
unicode-2-0-utf-8
UnicodeBig
UnicodeBigUnmarked
UnicodeLittle
UnicodeLittleUnmarked
us
US-ASCII
UTF-16
UTF-16,version=1
UTF-16,version=2
UTF16_BigEndian
UTF16_LittleEndian
UTF16_OppositeEndian
UTF16_PlatformEndian
UTF-16BE
UTF-16BE,version=1
UTF-16LE
UTF-16LE,version=1
UTF-32
UTF32_BigEndian
UTF32_LittleEndian
UTF32_OppositeEndian
UTF32_PlatformEndian
UTF-32BE
UTF-32LE
UTF-7
UTF-8
windows-10000
windows-10006
windows-10007
windows-10029
windows-10081
windows-1200
windows-1201
windows-1250
windows-1251
windows-1252
windows-1253
windows-1254
windows-1255
windows-1256
windows-1257
windows-1258
windows-20127
```

```
windows-20866
windows-21866
windows-28592
windows-28593
windows-28594
windows-28595
windows-28596
windows-28597
windows-28598
windows-28599
windows-28603
windows-28605
windows-31j
windows-437
windows-51949
windows-54936
windows-57002
windows-57003
windows-57004
windows-57005
windows-57006
windows-57007
windows-57008
windows-57009
windows-57010
windows-57011
windows-65000
windows-65001
windows-720
windows-737
windows-775
windows-850
windows-852
windows-855
windows-857
windows-858
windows-861
windows-862
windows-866
windows-869
windows-874
windows-874-2000
windows-932
windows-936
windows-936-2000
windows-949
windows-949-2000
windows-950
```

```
windows-950-2000
x11-compound-text
x-big5
x-compound-text
X-EUC-JP
x-IBM1006
x-IBM1025
x-IBM1097
x-IBM1098
x-IBM1112
x-IBM1122
x-IBM1123
x-IBM1124
x-IBM1153
x-IBM1363
x-IBM1363C
x-IBM1364
x-IBM1371
x-IBM1388
x-IBM1390
x-IBM1399
x-IBM33722
x-IBM33722A
x-IBM33722C
x-IBM720
x-IBM737
x-IBM856
x-IBM867
x-IBM874
x-IBM875
x-IBM921
x-IBM922
x-IBM930
x-IBM930A
x-IBM933
x-IBM935
x-IBM937
x-IBM939
x-IBM939A
x-IBM942
x-IBM942C
x-IBM943
x-IBM949
x-IBM949C
x-IBM950
x-IBM954
x-IBM954C
x-IBM964
```

```
x-IBM970
x-IBM971
x-ISCII91
x-iscii-as
x-iscii-be
x-iscii-de
x-iscii-gu
x-iscii-ka
x-iscii-ma
x-iscii-or
x-iscii-pa
x-iscii-ta
x-iscii-te
x-ISO-2022-CN-CNS
x-ISO-2022-CN-GB
x-iso-8859-11
x-ISO-8859-6S
x-JISAutoDetect
x-KSC5601
x-mac-ce
x-MacCentralEurope
x-mac-centraleurroman
x-MacCyrillic
x-mac-cyrillic
x-MacGreek
x-mac-greek
x-macroman
x-MacTurkish
x-mac-turkish
x-MacUkraine
x-MS932_0213
x-MS950-HKSCS
x-ms-cp932
x-roman8
x-sjis
x-UTF_8J
x-utf-16be
x-utf-16le
x-UTF-16LE-BOM
x-windows-1256S
x-windows-50220
x-windows-50221
x-windows-874
x-windows-950
x-windows-iso2022jp
```

Administrator Guide

Installing LightWave Server

These instructions apply to new installations. See Updating LightWave Server for instructions for updating an existing installation to a new release version. Note that you may install the product more than once (in separate subvolumes) on the same NonStop system if you wish.

Before proceeding with the installation process refer to the Release Notes for important information about the current release of the software, particularly the Installation Prerequisites.

The software is delivered in a single PAK archive containing the following files:

- SERVER - The LightWave Server process
- SWORKER - The LightWave Server Worker process
- LWSCOM - The LightWave Server Management CLI
- ICUDATA - Character set translation data.
- SCPnnnnn - The console installation package, where nnnnn is a version number
- ZSTARTUP - Sample SERVER process startup macro
- SUTILITY - The LightWave Server Utility process

SUTILITY has been deprecated and will no longer be updated. LWSCOM has replaced SUTILITY as the LightWave Server management CLI.

Installation Steps

- Obtain a Product License

Obtain a Product License

A license is required to run LightWave Server. If this is the first time you are installing the product on your NonStop system, you may obtain a free 30-day trial license at the Trial License Center. Otherwise, a license will be provided to you upon purchase of the product. Do not request a trial license until you are ready to begin using the product, as the 30-day timer starts on the day the license is issued. Note that you may install LightWave Server without a license, however the product will not function until a valid license is installed.

Download the Release Package

The software is distributed as a single PAK archive file. The software may be downloaded from our Software Download Center. Transfer the PAK file to your NonStop system using binary transfer mode.

Install the Release Package

Unpak the release package PAK archive file into an empty subvolume.

```
> UNPAK <pakfile> ($*.*.*), vol <installation-subvolume>, myid, listall
```

where <pakfile> is the name of the release package PAK archive file you transferred earlier.

Create a Filesystem

LightWave Server stores its operational data in a LightWave filesystem consisting of a collection of Enscribe files. A filesystem is created using the LWSCOM application as follows:

```
> run LWSCOM create filesystem <filesystem-subvolume>, password <admin-user-password>
```

where <filesystem-subvolume> is the name of a subvolume on a TMF-audited disk. The filesystem may be created in the same subvolume as the program files (the 'installation subvolume'), or a separate subvolume. The <admin-user-password> parameter specifies the password for the default "admin" user.

Optionally Install the Console Package

The LightWave Server Console provides a browser-based management interface to LightWave Server. Although the Console provides a convenient management interface, its use in some deployment scenarios may not be appropriate. When the Console is not installed, all management must be done using the LWSCOM program. In order to use the Console, install the console package as follows:

```
> run LWSCOM control filesystem <filesystem-subvolume>, installcon
<console-package-file>
```

where <filesystem-subvolume> is the name of a subvolume containing an existing LightWave Server filesystem. The <console-package-file> parameter specifies the the console package file included with the release in the form SCPnnnnn.

Install the Product License

Transfer the license file you received by email to your NonStop system using text or 'ascii' file transfer mode. The license must be stored in an 'edit' file in one of the following locations:

- the file LICENSE in the LightWave Server installation subvolume
- the file LICLWS in the LightWave Server installation subvolume
- the file $SYSTEM.NUWAVE.LICLWS

The LightWave Server SERVER process validates the license at startup and periodically while running. Any license errors will be written to the event log. Note that if the license file content is altered in any way, the license will become invalid. For more information see Product Licensing.

Start the Server Process

The LightWave Server SERVER process supports the user interface for the product. Refer to Starting LightWave Server for further instructions.

Once the SERVER process is running, open your browser to http://*system-name*:*port-number*, where *system-name* is your NonStop system name or IP address and *port-number* is the TCP/IP port number you used when starting the SERVER process. Login to the LightWave Server Console application using the username 'admin' and the password 'password' (or the new password you specified when you created the filesystem).

Updating LightWave Server

The process for updating LightWave Server from an earlier release to a later release is similar to the initial installation process. These instructions apply to 'full product' update releases. If you received a hotfix patch release, refer to the specific instructions included with the release.

Before proceeding with the update process refer to the Release Notes for important information about the current release of the software.

The software is delivered in a single PAK archive containing the following files:

- SERVER - The LightWave Server process
- SWORKER - The LightWave Server Worker process
- LWSCOM - The LightWave Server Management CLI
- ICUDATA - Character set translation data.
- SCPnnnnn - The console installation package, where nnnnn is a version number
- ZSTARTUP - Sample SERVER process startup macro
- SUTILITY - The LightWave Server Utility process

SUTILITY has been deprecated and will no longer be updated. LWSCOM has replaced SUTILITY as the LightWave Server management CLI.

Upgrade Steps

- Verify Your Product License
- Download the Release Package
- Shutdown Any Existing Processes
- Perform a Backup of the Existing Installation
- Install the Update Package
- Upgrade the Filesystem
- Optionally Install the Updated Console Package
- Restart Processes

Verify Your Product License

Product updates are available only to customers with a Support Agreement. Your product license file contains the expiration date of your Support Agreement. Each time you renew your Support Agreement, we issue you an updated license containing the new expiration date. View the contents of your product license file to ensure that your support expiration date is later than the release date of the product update you wish to install. See Product Licensing for information about the location of your product license. Contact Product Support if you have any question about your eligibility for installing updates. Regardless of your Support Agreement status, you *will* be able to download and install the update. However, **the product will not run if your license support expiration date is earlier than the release date of the product version you are installing**.

Download the Release Package

The software is distributed as a single PAK archive file. The software may be downloaded from our Software Download Center. Transfer the PAK file to your NonStop system using binary transfer mode.

Shutdown Any Existing Processes

Ensure that there are no active users of applications which depend on LightWave Server. From the existing LightWave Server installation subvolume, use the FILEINFO command to ensure that no files are open. Stop any processes that have files opened. From the LightWave Server installation subvolume:

```
> run server --shutdown <server-process-name> !
> status *, prog SERVER, stop
```

Perform a Backup of the Existing Installation

Verify that you have a current backup of the existing installation, including the LightWave Server filesystem (FS0* files). In the unlikely event of an unsuccessful update, you will need to restore the installation from this backup. You can use BACKUP, or use the PAK utility to perform the backup if you like. If you created the filesystem in a subvolume other than the installation subvolume, be sure to backup both subvolumes.

```
> PAK <pakfile>, (*), audited, listall
```

where <pakfile> is the name of the backup PAK archive file to be created, which should be in a separate subvolume.

Install the Update Package

We recommend that you install the update in the same subvolume as the existing installation in order to eliminate confusion over the location of the "current" version. However, it is possible (i.e. supported), to install the update into a new subvolume and even to run different versions in parallel, but bear in mind that LightWave Server filesystems (FS0* files) cannot be shared between installations.

Verify that you have the necessary permissions to overwrite the exiting files. Unpak the release package PAK archive file, overlaying the originally installed files.

```
> UNPAK <pakfile> ($*.*.*), vol <installation-subvolume>, myid, listall
```

where <pakfile> is the name of the release package PAK archive file you transferred earlier.

Upgrade the Filesystem

LightWave Server stores its operational data in a LightWave filesystem consisting of a collection of Enscribe files. The filesystem must be upgraded to incorporate any structural or content modifications included in the new release. The filesystem is upgraded using the LWSCOM program, which is run as follows:

```
> run LWSCOM control filesystem <filesystem-subvolume>, upgrade
```

where <filesystem-subvolume> is the name of the subvolume that contains the LightWave Server filesystem.

Optionally Install the Updated Console Package

The LightWave Server Console provides a browser-based management interface to LightWave Server. If the filesystem has an existing Console installation, then the console package file included with the new release must be installed. Install the console package as follows:

```
> run LWSCOM control filesystem <filesystem-subvolume>, installcon <console-package-file>
```

where <filesystem-subvolume> is the name of a subvolume containing an existing LightWave Server filesystem. The <console-package-file> parameter specifies the console package file included with the release in the form SCPnnnnn.

Restart Processes

Restart the LightWave Server components as per your locally-developed procedures.

Starting LightWave Server

LightWave Server runs as a group of NonStop Server Guardian processes using the Supervisor-Worker pattern. The SERVER process services the management console and connection requests for API services. When a client creates a TCP/IP connection to a LightWave Server TCP/IP port, the SERVER process delegates the connection and all future processing of that connection to an SWORKER process. In the event that no SWORKER processes are attached to the SERVER process, the SERVER process itself takes over processing of API service requests. This allows LightWave Server to easily scale according to demand.

The SERVER process runs as a single fault tolerant process pair and may be configured to listen on one or more TCP/IP processes and ports. To meet service demand, any number of SWORKER processes, running in any CPU, can be attached to a supervisor. Worker processes assume control of TCP/IP connections from the SERVER but do not listen on TCP/IP ports and therefore will not conflict with other processes listening on TCP/IP ports.

Tip

A sample LightWave Server startup macro, ZSTARTUP, is included with the release. Before using for the first time, copy or rename the file to a new name (e.g. FUP DUP ZSTARTUP, STARTUP) so it won't be overwritten by future releases. The sample macro should be suitable for most installations, but it may be necessary to modify certain parameters for your environment. To use the macro: RUN STARTUP

Starting a Server Process

LightWave Server is started by running the SERVER program from TACL.

```
tacl> run server / run-options / command-line-options
```

Starting a Worker Process

An Worker process is started by running the SWORKER program from TACL.

```
tacl> run sworker / run-options / --server $server-process-name
```

More Information

Refer to the Command Line Reference for details about starting LightWave Server and Worker processes.

Examples

Start the SERVER process as a process pair in CPU 0 and 1 with SWORKER processes in CPU 2 and 3. The Console may be accessed on port 8080 using HTTP or port 8443 using HTTPS. Services may be accessed on port 9080 using HTTP or 9443 using HTTPS. This example assumes that a server certificate with Common Name *lightwave.example.com* has already been installed.

```
tacl> run server / name $LWS, cpu 0 / --backupcpu 1 &
--console-ports $ztc0:8080 $ztc0:8443:lightwave.example.com &
--service-ports $ztc0:9080 $ztc0:9443:lightwave.example.com &
--log $zhome info
tacl> run sworker / name, cpu 2 / --server $LWS
tacl >run sworker / name, cpu 3 / --server $LWS
```

Stopping LightWave Server

A LightWave Server instance may be shut down with the SERVER –shutdown command using the following syntax:

```
tacl> run SERVER --shutdown <process-name> [!]
```

where <process-name> is the name of the SERVER or SWORKER process to shut down. If "!" is specified, the shutdown will occur immediately with all current requests abruptly terminated.

This command may be used to shut down the SERVER process or individual SWORKER processes. When a SERVER process is shut down, all connected SWORKER processes are also shut down. The default behavior is to allow connections to drain, i.e., the SERVER will stop accepting connections but will allow existing connections to continue being serviced until they are all disconnected. Using the "!" option will cause the target process to immediately terminate all connections and shut down.

Note that while draining, the SERVER process will not shut down until all Console connections are terminated.

Examples

Stop the supervisor and all attached workers.

```
tacl> run server --shutdown $lws
```

Managing LightWave Server

The LightWave Server Console provides access to configuration and management functions and developer tools. The Console is accessed using any supported browser to navigate to the console port of a LightWave Server instance. Note that you must connect to the port that has been configured using the –console-ports option. For example, if your LightWave Server instance was started using the following command:

```
tacl> run SERVER / name $lws, cpu 0 / --backupcpu 1 --console-ports $ztc0:8080 --
service-ports $ztc0:8090 --log $zhome info
```

Then the URL for the console is:

```
http://host-name:8080
```

Console Features

The following sections describe features available in the Console.

- Signing in to the Console
- The Dashboard
- Dictionaries
- APIs
- Access Control Policies
- Services
- HTTP Logs
- Diagnostic Logs
- Server Certificates
- Users and Groups
- Managing the Filesystem
- Using Configuration Files

The LightWave Server Console provides access to configuration and management functions and developer tools. The Console is accessed using any supported browser to navigate to the console port of a LightWave Server instance. Note that you must connect to the port that has been configured using the --console-ports option. For example, if your LightWave Server instance was started using the following command:

```
tacl> run SERVER / name $lws, cpu 0 / --backupcpu 1 --console-ports $ztc0:8080 --
service-ports $ztc0:8090 --log $zhome info
```

Then the URL for the console is:

```
http://host-name:8080
```

Console Features

The following sections describe features available in the Console.

- Signing in to the Console
- The Dashboard
- Dictionaries
- APIs
- Access Control Policies
- Services
- HTTP Logs
- Diagnostic Logs
- Server Certificates
- Users and Groups
- Managing the Filesystem
- Using Configuration Files

Signing in to the Console

When you navigate to the Console URL, a Sign In page is presented. Enter your user ID and password to sign in. Note that:

- If this is a new LightWave Server installation, the default user ID is "admin", with password "password". The default password should be changed immediately after logging in for the first time.
- Only users that are enabled for Console access may log into the Console.

After successful log in, the Console dashboard is displayed. You may navigate to other areas of the Console using the menu on the left side of the page. If the browser window size is reduced, the menu will auto-hide to provide the largest viewing area for the Console window. When the menu is hidden it may be opened by clicking the menu icon ≡ in the upper left corner of the page.

The Dashboard

The Dashboard view provides status information for your LightWave Server instance. The Dashboard contains three sections.

Processes

This section shows the current status of the LightWave SERVER and SWORKER processes the make up your LightWave Server instance. The view shows the process names, program file locations, CPUs and the current number of TCP/IP connections for each process

Ports

This section shows the TCP/IP ports that the SERVER process is listening on. The view shows the TCP/IP process name, the port, the protocol and the available services on each port. When the Protocol is HTTPS, the Common Name of the server certificate for the port is displayed. HTTPS ports may be used for TLS connections.

Filesystem

This section shows the location of the filesystem and the percentage of free space available. Because the majority of filesystem space is consumed by HTTP and Diagnostic logs, the status of those logs is also displayed.

License

This section shows the version of the running processes and the license status. For more information see Product Licensing.

Dictionaries

The Dictionaries view provides for management of application data type dictionaries.

- Dictionary List View
- Dictionary Detail View

Dictionary List View

The Dictionary list view shows the dictionaries that are currently installed in the Dictionary Repository. If you have a large number of Dictionaries, the filter can be used to reduce the number of Dictionaries listed by entering a partial name of the dictionaries that you wish to view.

You may view the details of a Dictionary by selecting the Dictionary name to open the Dictionary.

Create a New Dictionary

Dictionaries may be imported from existing NonStop Server DDL or DDL2 dictionaries.

To create a new Dictionary:

1. Click the + icon to open the Create Dictionary dialog.
2. Enter the name of the new Dictionary.
3. Enter the name of the Guardian subvolume that contains the DDL dictionary.
4. By default, all objects in the DDL dictionary are imported into the LightWave dictionary. If appropriate, select a subset of the objects to import. Note that the dependencies of an object need not be individually selected, they are automatically imported.
5. Click Create.

Importing a Dictionary

You can import a dictionary from a previously exported or manually created dictionary file.

To import a previously exported Dictionary file:

1. Click the ⬆ icon to open the upload dialog.
2. Click the ⬆ to open a file browser and select the Dictionary file.
3. Enter the name of the imported Dictionary.
4. If appropriate, select the checkbox to replace an existing dictionary with the same name.
5. Click Upload.

Dictionary Detail View

The Dictionary Detail view shows the types available in the dictionary and provides the ability to modify the dictionary. If you have a large number of types in the dictionary, the filter can be used to reduce the number of Dictionaries listed by entering a partial name of the types that you wish to view.

Type Information

You can view the type information detail for each type in the dictionary in JSON or binary format.

To view the Type information:

1. Click the type name to open the type view.
2. To view a sample JSON representation of the type, click the JSON tab.
3. To view the binary layout of the type click the Binary tab. The Binary view is similar to a DDL dictionary report.

Renaming a Dictionary

Note that renaming a dictionary may impact user-defined API definitions or native API applications that reference the dictionary name.

To rename the Dictionary:

1. Click the ✏ icon to open dictionary rename dialog.
2. Enter the new name.
3. Click Rename to complete the rename or Cancel to exit without renaming.

Refreshing a Dictionary

If the DDL dictionary that a LightWave Dictionary is based on has changed, you may wish to refresh the LightWave dictionary from the source DDL.

To refresh the Dictionary from source DDL:

1. Click the ↻ icon to open dictionary refresh dialog.
2. By default only existing objects in the LightWave dictionary are refreshed. If new objects exist in the DDL dictionary, they may be selected for import.
3. Click Refresh to complete the refresh or Cancel to exit without refreshing.

Exporting a Dictionary

Exporting the Dictionary allows you to create a backup copy of the Dictionary or import it into another LightWave Server instance.

To export the dictionary to a file:

1. Click the ⤓ icon to open the system file browser.
2. Select a download location for the file and click Save to download the Dictionary to a file.

Deleting a Dictionary

Dictionary deletion is not reversible. Note that when APIs are deployed as Services a snapshot of the API and related dictionaries are saved with the Service definition, however, APIs available in the API Editor will no longer function if the dictionaries they reference are deleted. **Before deleting a dictionary make certain that is is not being referenced by an API.** If you think you may need a copy of the dictionary at a later time, export the dictionary before deleting it.

To delete the Dictionary:

1. Click the 🗑 icon to open dictionary delete dialog.
2. Click Delete to complete the deletion or Cancel to exit without deleting.

APIs

The APIs view provides for management of User-defined APIs. For more information see Working with User-Defined APIs in the Developer Guide.

Access Control Policies

An Access Control Policy (ACP) determines which users are granted access a service. An ACP is comprised of access control rules, of which there are three types; Identity Rules, Source IP Rules, and CORS Origin Rules.

Identity Rules

An identity rule grants access to a specific user or group. An ACP may contain zero or more identity rules. If no identity rules are defined, access is granted to all users – so called 'anonymous access'. If more than one identity rule is defined, the rules are logically OR-ed (for example, to allow access to a user who is a member of group Accounting OR group Engineering OR is user Operator). This is one special identity rule that can be defined: user "Any authenticated user", which means access is granted to any user that supplies a valid user name and password. 'Any authenticated user' is different from 'anonymous access' in that in the latter case, no user name or password is required.

Source IP Rules

A source IP address rule grants access to applications running on devices with specified IP addresses. An ACP may contain zero or more source IP address rules. In no source IP address rules are defined, access is granted to all source IP addresses. If more than one source IP address rule is defined, the rules are logically OR-ed. A source IP address can be a complete IP address (192.168.168.168, for example) or a range of address in CIDR notation (e.g. 172.168.168.0/24).

The set of identity rules and source IP address rules are logically AND-ed to determine access under the policy. So, given an ACP with the rules (group Accounting OR group Engineering OR user Operator) AND (source IP 192.168.168.168 OR source IP 10.1.0.0/16), user 'jsmith', who happens to be a member of the Accounting group, will be denied access if his application is running on a device with IP address 172.100.0.96, but allowed access if running on a device with IP address 10.1.224.17.

CORS Origin Rules

Cross Origin Resource Sharing (CORS) rules provide a means to restrict browser cross origin requests to a specific origin or origins. Origins are specified in the format specified by the CORS standard. For more information see the CORS specification or learn more about CORS.

Creating an Access Control Policy

Access Control Policies are managed through the LightWave Server Console.

To create a new Access Control Policy:

1. Select "Access Control" from the menu.
2. Click the + icon to add a new policy.
3. Enter a name and description.
4. Add identity source IP address, or CORS rules by clicking the + icon on their respective toolbars.
5. Save the policy.
6. Changes to policies are effective immediately.

Services

Services make APIs accessible to users. A service, through its assigned Access Control Policy, determines which users are allowed access. Furthermore, a service can limit which server TCP/IP ports will be used for the service. A service is comprised of a single API.

Services are managed through the LightWave Server Console. After you have defined an API and created an Access Control Policy, you can deploy them as a Service.

> When a service is deployed, a snapshot is taken of the API definition and any dictionaries associated with the API. This allows the original API definition and dictionaries to be modified without affecting the deployed service.

To deploy a new service:

1. Select "Services" from the console menu.
2. Click the + icon to add (deploy) a new service.
3. Enter a name and description for the service.
4. Supply a version identifier for the service.
5. Select the API to be exposed by the service.
6. Select the appropriate Access Control Policy from the list.
7. If desired, restrict the TCP/IP ports at which API requests will be accepted.
8. Enable the service and save. The API associated with the service is immediately available.

To redeploy a service:

Redeploying a service refreshes the service from the origin API definition and dictionaries.

1. Select "Services" from the console menu.
2. Select the Service you wish to redeploy.
3. Click the ↻ icon to open refresh confirmation dialog.
4. Click OK to complete the refresh or Cancel to exit without refreshing.

To export a service:

Exporting a service creates a service package file that contains the API and all related dictionaries for a service. The service package can then be imported into another server instance for deployment.

1. Select "Services" from the console menu.

2. Select the Service you wish to export.
3. Click the ⬇ icon to open the system file browser.
4. Select a download location for the file and click Save to download the Service package file.

To import a Service:

1. Click the ⬆ icon to open the upload dialog.
2. Click the ⬆ to open a file browser and select the Service package file.
3. Enter the name of the Service.
4. If appropriate, select the checkbox to replace an existing Service with the same name.
5. Click Upload.
6. Complete the service configuration by selecting an Access Control Policy and optionally enabling the Service.

HTTP Logs

HTTP Logs show HTTP request traffic to your LightWave Server instance. HTTP logs are formatted in standard NCSA Common log format. If you have a large number of logs, the filter can be used to reduce the number of logs listed by entering a partial name of the log that you wish to view.

HTTP Logging is disabled by default.

To enable or disable logging:

1. Click the ⚙ icon to open the HTTP Log Settings dialog.
2. Select or deselect HTTP Logging
3. Select or deselect *Include Server Console traffic in the log*
4. Click Save to update the settings or Cancel to exit without updating.

The LightWave Server Console creates a large volume of HTTP traffic to the SERVER process which does not represent client API requests. Deselecting this traffic can reduce the size of your HTTP logs.

To delete one or more logs:

1. Select one or more log entries or click the ✓✓ icon to select all logs.
2. Click the 🗑 icon to open the delete confirmation dialog.
3. Click Delete to delete the selected logs or Cancel to exit without deleting.

To download a selected log

1. From the log detail view click the ⭳ icon to open the system file browser.
2. Select a download location for the file and click Save to download the log to a file.

Diagnostic Logs

Diagnostic Logs show detailed information about API requests to your LightWave Server instance. If you have a large number of logs, the filter can be used to reduce the number of logs listed by entering a partial name of the log that you wish to view. Logs are prefixed with the service name that generated them.

> ℹ Diagnostic logging is resource intensive and will degrade the performance of all services on your LightWave Server instance. Diagnostic logging should not be enabled on production instances unless necessary. Use of the *Auto-disable timer* option is strongly recommended to ensure that Diagnostic logging is not left enabled indefinitely.

Diagnostic Logging is disabled by default. To manage Diagnostic Logging, select *Diagnostic Logging* from the sidebar menu.

To enable or disable logging:

1. Click the ⚙ icon to open the Log Settings dialog.
2. Select or deselect Enable Logging
3. If enabling logging, set the auto disable timer to automatically disable logging after a defined number of minutes.
4. If desired, enter an HTTP status code filter range. Only requests with status codes in the specified range will be logged.
5. If desired, configure Log Content Selection. Excluding the HTTP body or server IPM can reduce the size of the log when this information is not of interest.
6. Select the services for logging from the list of deployed and enabled services.
7. Click Save to update the settings or Cancel to exit without updating.

To delete one or more logs:

1. Select one or more log entries or click the ✓✓ icon to select all logs.

2. Click the 🗑 icon to open the delete confirmation dialog.
3. Click Delete to delete the selected logs or Cancel to exit without deleting.

To download a selected log

1. From the log detail view click the ⤓ icon to open the system file browser.
2. Select a download location for the file and click Save to download the log to a file.

Request Processing Time

The diagnostic log contains request processing time information. This table describes the timing information:

Time	Description
Connect	The time between the acceptance of the socket connection and the completion of the TLS handshake, if any. If a connection was reused, the Connect time will be "N/A".
Request	The time between completion of the connection and the completion of reception of the entire web service request from the network.
Deserialize	The time between completion of request reception and the completion of deserialization of the web service request headers and payload into the application server request IPM.
Server I/O	The time taken by the application server I/O. For process servers, this includes the time for server OPEN, if necessary.
Serialize	The time between the completion of the server I/O and the completion of the serialization of the server reply IPM into the web service response headers and payload.

Time	Description
Response	The time between completion of serialization and the completion of the network send of the request. Note that although the completion of the send is indicated by the NonStop TCP/IP process, it does not necessarily mean that the web service client has received the entire response. Delays in the network, including one or more proxies, may increase the time required for the web service client to receive the entire response.
Total	The total time between receiving the request from and returning the response to the web service client.

Server Certificates

The Server Certificates view provides for the management of X509 certificates used for HTTPS connections. Three types of certificate resources may be managed:

- Certificate Signing Requests - A Certificate Signing Requests (CSR) is a request for a new certificate which will be forwarded to a Certificate Authority (CA) for signing. The CSR contains the identity parameters for the new certificate.
- Server Certificates - Server Certificates are X509 certificates that have been signed by a trusted Certificate Authority. When a CSR is submitted to a CA, a signed certificate is returned and installed in LightWave Server. Server Certificates may be used to configure HTTPS Ports by specifying the certificate Common Name (CN) in the Port configuration.
- Intermediate Certificates - Intermediate Certificates are certificates that help establish the trust chain between a Server Certificate and the CA that signed it. Intermediate Certificates, if necessary, are supplied by the CA.

The process of requesting a new certificate and installing it in the server is as follows:

- Create a Certificate Signing Request
- Submit the CSR to a Certificate Authority for signing
- The CA will optionally verify the identity information in the CSR and generate a signed certificate.
- The CA will return the signed certificate with any necessary Intermediate Certificates
- The signed certificate and Intermediate Certificates are installed.
- The certificate my now be used to configure HTTPS ports.

A trusted HTTPS connection requires a Server Certificate signed by a recognized Certificate Authority. For testing purposes, LightWave Server provides for generation of self signed certificates which are immediately valid for use as Server Certificates. You may also use free Certificate Signing services such as getaCert to test the certificate signing process. Although useful for testing, these certificates will result in certificates verification errors when used and should never be used for production service.

Create a Certificate Signing Request

Begin the CSR creation process by selecting the + icon in the **Certificate Signing Requests** toolbar. Complete the dialog with identity values that are valid for your organization.

Common Name	Enter the fully qualified host name of your server, for example, www.example.com
Alternate Host Names	Enter a comma separated list of alternate host names for your server. Note that your CA may not support this feature.
Country Code	Select country from the drop down list.
State or Province	Enter the state or province of your organization. Abbreviations should be avoided.
City	Enter the locality name of your organization.
Organization	Enter the name of your organization. This field is optional
Organizational	Enter the name of the unit within your organization. This field is optional

Select the self signed certificate option if you wish to generate and install a self signed certificate. When the **Create** button is selected the new CSR or self-signed certificate is displayed. If a CSR was created, you may select and copy the CSR content from the display, or download the CSR as a file by selecting the ⤓ icon. The CSR may now be forwarded to your Certificate Authority. Note that LightWave Server only supports PEM (base64 encoded) format certificates.

Install the Signed Certificate

To install the signed certificate returned by the CA, open the associated CSR and select the ✓ icon. Paste the entire content of the signed certificate into the **PEM Format Certificate** field and select **Install**. The certificate will be installed and displayed and the CSR will be removed.

Install Intermediate Certificates

If your CA provides Intermediate Certificates, they may be installed by selecting the + icon in the Intermediate Certificates toolbar. Paste the entire content of the Intermediate Certificate(s) into the **PEM Format Certificate** field and select **Install**. Multiple certificates may be installed at once. LightWave Server determines the application and correct order of the Intermediate Certificates so the certificates may be installed in any order.

Using Certificates

Once installed, a certificate may be used to enable the HTTPS protocol on LightWave SERVER console and/or service ports. The ports must be configured at SERVER startup. Use the Common Name (see above) when specifying the port. See SERVER Command Line Options for more information about configuring TCP/IP ports.

```
tacl> run server --console-ports 80 443:www.example.com --service-ports 8080
8443:www.example.com
```

Users and Groups

The User and Group management features allows you to create and manage LightWave Server users and groups. Users and Groups may then be used with Access Control Policies to restrict access to API services.

To add a new user:

1. Select "Users" from the Console menu.
2. Click the + icon.
3. Enter the user name, which is also the users sign-in user ID, and the user's full name.
4. Enter the user's password.
5. Select the groups of which the user is a member. Begin typing in the group field to select the group by name or enter a period to show a list of all groups.
6. Select whether or not the user has access to the Server Console
7. Select whether or not the user is disabled.
8. Click Save to save the user or Cancel to exit without saving.

To add a new group:

1. Select "Groups" from the Console menu.

2. Click the + icon
3. Enter the group name and a group description.
4. Select the users that are members of the group. Begin typing in the user field to select the user by name or enter a period to show a list of all users.
5. Click Save to save the group or Cancel to exit without saving.

Managing the Filesystem

The LightWave Server filesystem is the repository for all LightWave Server configuration data. It contains API and Service definitions, Access Control Policies, Server Certificates, configured users and groups, and HTTP and Diagnostic Logs. The filesystem consists of four audited Enscribe files: FS00, FS00A, FS01 and FS02. A filesystem cannot be shared by more than one LightWave Server SERVER process. The LWSCOM utility is used for Filesystem management.

Copying or Moving the Filesystem

You copy the LightWave Server filesystem by copying all of the FS0* files in the filesystem subvolume to a new subvolume. The preferred way to do this is by using the PAK and UNPAK utilities (or similarly BACKUP and RESTORE) since they automatically update alternate key file references. A move is a copy to a new location, followed by a delete of the original.

Important

Before attempting to copy or move the filesystem, make sure its files are not open.

```
tacl> fup listopens $DATA.MYFS.FS0*
```

Copy the FS files in subvolume $DATA.MYFS to $DATA.MYNEWFS.

```
tacl> pak myfspak, $DATA.MYFS.FS0*, audited, listall
tacl> unpak myfspak, $*.*.*, audited, listall, myid, map names ($DATA.MYFS.* to
$DATA.MYNEWFS.*)
```

Alternatively, you can use FUP DUP to copy the files, but you *must manually update the alternate key file references*.

```
tacl> volume $DATA.MYNEWFS
tacl> fup
- dup $DATA.MYFS.FS0*, *, saveall
- alter FS00, altfile (0, FS00A), audit
- alter FS00A, audit
- alter FS01, audit
- alter FS02, audit
- exit
```

Specify the new location of the filesystem using the --filesystem command line argument when starting the SERVER process. Once you have verified that you have successfully copied the filesystem to its new location, you may delete the original if desired.

If you are copying to another NonStop system, note that the filesystem contains one or more server certificates (if installed) which contain information specific to the host name used to access LightWave Server. You may need to obtain a new server certificate for your additional system.

Reducing Filesystem Disk Space Usage

When files are deleted from the filesystem the files are only marked for deletion. The file data records are physically deleted from the FS00 & FS01 files over time by a housekeeping process within LightWave Server. You can use LWSCOM to cause the FS00 & FS01 data records to be immediately deleted by running the following command from the TACL prompt:

```
tacl> run lwscom control filesystem <filesystem-subvol>, clean
```

After cleaning the filesystem there may still be a large amount of unused disk space allocated to the FS01 file and the Console may report that the filesystem is full or nearly full. This is due to the way that data records are organized in the file. While there may be many free blocks available, they are interspersed with in-use blocks and cannot be released by Enscribe. You can reclaim this disk space using the FUP RELOAD command by running the following command from the TACL prompt:

```
tacl> fup reload <filesystem-subvol>.fs01
```

This creates a nowait process ORSERV which reorganizes the file data records and releases the unused space. This process can be run while LightWave Server is running. Note that you may need to run this command multiple times before all of the free space is reclaimed.

Using Configuration Files

Some process configuration options support dynamic configuration through the use of configuration files. Instead of providing the configuration options on the process startup command line, the configuration file name is supplied and the configuration options are read from the file. The –monitor option can be used to enable change monitoring for the configuration files. When changes are detected the options are reloaded from the file. Configuration files are Enscribe EDIT files and use the "INI" file format. The following commands may use configuration files:

Command	Description
--log	Logging configuration

✓ When monitoring is enabled, the file is continuously monitored for changes. When a change occurs, the file is reloaded and the new options take effect. If the configuration change contains an error or the file is inaccessible then the change is ignored and a notification message is output the the log file.

The following sections describe the configuration options for these commands:

- Configuration File Format
- Log Configuration

Configuration File Format

Configuration files are Enscribe EDIT files in the "INI" file format.

Options

Options consist of name-value pairs delimited by an equals sign (=). The value may be enclosed in quotes to preserve whitespace when necessary.

Sections

Parameters are grouped into named sections. A section header consists of the a section name enclosed in square brackets ([ ]). All parameters found after a section header and before the next section header or end of file are associated with that section header.

Section headers may include a process name to indicate that the section applies to a specific process. In this case the section name consists of the process name and section name delimited by a colon (:).

Comments

A hash (#) character indicates the start of a comment, which continues to the end of the line. All text between the hash and end of line is ignored.

Blank Lines

Blank lines are ignored.

Example

```
# This section configures generic logging options
[log]
file=zzlog
level=info   # log at info level.
format=text
```

Log Configuration

Dynamic logging configuration is activated using the --log startup option and specifying the log configuration file location. *Note that the file name specification is preceded by a '+' character.*

```
run SERVER --log +$vol.subvol.logcfg [--monitor log:15 ] ...
```

Configuration Reference

The log configuration must be preceded by a *[log]* header

Option	Descriptions
file	The name of the log file. If not fully qualified, then the value of the _DEFAULTS define is used to complete the file name.

Option	Descriptions
level	The level value may be "error", "warning", "info", or "debug" and controls the type of information that is output to the log destination. The "error" level produces the least output while the "debug" level produces the most output. The default value is "info".
format	The format value may be "text" indicating that the log events should be output as text strings or "event" indicating that the log events should be output in EMS event format. If not specified, the default value is "text".

Examples

```
# Log to a file
[log]
file=$data1.logs.zzlog
format=text
level=info
```

```
# Log to $0
[log]
file=$0
format=event
level=info
```

Configuration Best Practices

Due to the tremendous variation in resource demands of customer applications and available system resources, we cannot provide specific configuration recommendations. However, below are some suggestions that you should consider when planning your configuration. You should test your application under simulated production conditions to validate your configurations prior to deployment.

Performance & Scalability

- Run SERVER as a process pair (use the --backupcpu option).
- Run at least one SWORKER process.
- Start as many SWORKERs as needed according to anticipated load, each in different processors.
- The SERVER process distributes load across available SWORKERs to the SWORKER with the least number of connections
  - Consider setting --http-keepalive-timeout (default 30s) to limit connection time length.
  - Consider settting --http-keepalive-max (default 100) to close connections after this number of requests.
- If large numbers of TCP/IP connections are anticipated, configure multiple --service-ports using separate TCP/IP processes.
- If TMF transactions are being used (by client applications or in API definitions).
  - Consider setting --max-tx-timeout to set an absolute limit on TMF transaction timeouts.
  - Consider setting --default tx-timeout to set the default TMF transaction timeout if not specified by client or API.
- Leave HTTP logging disabled (installation default).
- Use the monitor option where appropriate, to monitor changes to configuration files. See --monitor.
- Do not use diagnostic logging in performance sensitive environments unless absolutely necessary.

Security

- Change the default administrator password!
- Use Server Certificates and configure only HTTPS console and service ports.
- Disable TLS v1.0 and v1.1 if possible. See --tls-protocols.
- Use restrictive Access Control Policies.

- Consider an external security appliance or reverse proxy if connected to untrusted networks.
- Maintain separate development and production LightWave Server instances.
- Use the *sensitive* schema property to avoid disclosing sensitive data in logs. See Sensitive Data Masking.
- Leave the 'lightwave-api-testing-service' disabled (installation default) in production instances.

Command Line Reference

- LWSCOM
- SERVER
- SUTIILITY
- SWORKER

LWSCOM

The LWSCOM command line interface (CLI) can be used as an alternative to the LightWave Server Console for management of a LightWave Server instance. Commands may be entered interactivity or supplied with a TACL macro or obey file. LWSCOM supports the following commands and objects:

Commands

!	ADD	ALLOW	ALTER
CONTROL	CREATE	DELETE	ENV
EXIT	EXPORT	FC	FILESYSTEM
HELP	HISTORY	IMPORT	INFO
OBEY	PAGESIZE	STATUS	VALIDATE
VOLUME			

Objects

ACP	API	CERTIFICATE	CONFIG

DIAGLOG	DICTIONARY	FILESYSTEM	GROUP
HTTPLOG	LICENSE	SERVICE	USER

LWSCOM is self-documenting using the HELP command. For example:

```
TACL> lwscom
LightWave Server COM 1.1.7 - \NODE - $XYZ
Copyright (c) 2024 NuWave Technologies, Inc. All rights reserved.
LWSCOM 1-> help
Help is available on the following commands and objects:
Commands:

!                  ADD                ALLOW              ALTER
CONTROL            CREATE             DELETE             ENV
EXIT               EXPORT             FC                 FILESYSTEM
HELP               HISTORY            IMPORT             INFO
OBEY               PAGESIZE           STATUS             VALIDATE
VOLUME

Objects:

ACP                API                CERTIFICATE        CONFIG
DIAGLOG            DICTIONARY         FILESYSTEM         GROUP
HTTPLOG            LICENSE            SERVICE            USER

Enter HELP <command> | <object> for more information.
LWSCOM 2-> help acp
The following commands can be used with the ACP object:

ADD                ALTER              DELETE             INFO

Enter HELP <command> ACP for more information.
LWSCOM 3-> help all acp

ALLOW Command

Specifies the maximum number of warnings or errors that are allowed to occur
before execution of an OBEY file or IN file is terminated.

  ALLOW [ [ <count> | ALL | NO ] [ ERRORS | WARNINGS ] ]

[ <count> | ALL | NO ] [ ERRORS | WARNINGS ]

  ALL indicates that there is no limit on the number of errors or warnings. NO
  indicates that no errors or warnings are allowed. <count> is an integer
  specifying the number or errors or warnings. If the ALLOW command is used
  with no parameters, the default setting is NO ERRORS and ALL WARNINGS. If
  ERRORS or WARNINGS is specified but ALL, NO, and <count> are omitted, ALL is
  assumed. IF ALL, NO, or <count> is specified, but ERRORS and WARNINGS are
  omitted, ERRORS is assumed.

LWSCOM 4->
```

SERVER

SERVER is the LightWave Server process. It supports the management console and connection requests for API services.

Starting the Process

The SERVER process is started by running the SERVER program from TACL.

```
tacl > run SERVER / run-options / command-line-options
```

You should be logged-on as a user with sufficient privileges to access the system resources that the process requires.

run-options

The standard TACL run options. The 'CPU' option is recommended if the -backupcpu command-line-option is specified. The NOWAIT option is recommended. The TERM option is also recommended if started from a dynamic terminal device. The IN and OUT options are ignored.

command-line-options

@<command-file>

> Reads command line options from <command-file>. Options specified on the command line override any duplicates specified in the file. At most, one '@' option may used. The file itself cannot contain an '@' option (i.e., no nesting).

--allow-shared-filesystem

> When present, one or more SERVER processes may share a single filesystem. If omitted, only one server process and its backup process are able to access a single filesystem.

--backupcpu

Specifies the number of the CPU in which the process should run its backup process. It must not be the same as the primary CPU. If omitted, no backup process is started.

--blob-files [$vol].subvol].file-name-prefix [userid,groupid | groupname,username] [security-string] [extents=<pri>,<sec>,<max>]

A pattern which specifies the file system location and file name prefix for output BLOB files and optionally, the user id and file security. The file name prefix is limited to 1 to 3 characters with the remaining 5 characters assigned by the SERVER / SWORKER process. If the volume or subvolume portion of the pattern is omitted, the process default volume and subvolume are used. If the option is omitted, the default pattern is "$current-vol.current-subvol.BLB" and the userid and file security are that of the SERVER / SWORKER process. Note that client applications are responsible for disposing of the BLOB files once they have been processed.

--console-ports <port-specification> [<port-specification>] ...

Specifies a list of TCP/IP ports that the process should listen on for browser connections. <port-specification> has the following form:

- [<tcpip-process>:]<port-number>[:<certificate-name>][:bindaddr=<ip-address>]

where:

- <tcpip-process> - The NonStop TCP/IP process to use for this port. If omitted, the value of the define =TCPIP^PROCESS^NAME is used if it exists, otherwise $ZTC0.
- <port-number> - The TCP/IP port to listen on, 0-65535. Port numbers 0-1023 can only be used if run by a member of the SUPER group
- <certificate-name> - The CN value of the X509 certificate to use for this port. If specified, the port will use HTTPS protocol.
- <ip-address> - The IP address to bind to when the TCP/IP provider has multiple IP addresses configured.

Multiple port specifications are separated by spaces. If this option is omitted, no console ports will be opened. See Server Certificates for information about installing server certificates.

--default-encoding <encoding-name>

Specified the default encoding to use for character string conversions in all services. Note that encoding settings applied to API method or data type definitions will override this setting. The <encoding-name> must be one of the names listed in Character Encoding Names. If omitted, the default encoding is ISO-8859-1.

--default-tx-timeout <seconds>

Specifies the default transaction timeout value for transactions started by client applications when the lw-transaction-timeout header is not included in the request.

The value must be in the range 0 to 21474836. If not specified, the value of --max-tx-timeout is used as the default.

--disable-digest-auth

Disables HTTP Digest authentication and prevents the server from advertising that Digest authentication is available through the WWW-Authenticate header.

--disable-icu

Disables the ICU package which is used for character string encoding. When disabled, character string encoding is limited to ISO-8859-1.

--disable-sensitive-data-masking

When present, the sensitive data masking feature is disabled and fields marked as sensitive will be displayed in HTTP and diagnostic logs. This option should only be used during application development when sensitive data is not contained in message payloads.

--filesystem <subvolume-spec>

The location ($volume.subvolume) that contains the LightWave File System. If omitted, the default (current) subvolume is searched, then the installation subvolume (containing the SERVER program file) is searched. Note that there is a one-to-one correspondence between each SERVER and a given filesystem; filesystems may not be shared among multiple instances of SERVER. Refer to Installing LightWave Server for information about creating the file system.

--http-keepalive-max <count>

Specifies the maximum number of requests that may occur on a persistent connection before the server closes it. The default value is 100.

--http-keepalive-timeout <seconds>

Specifies the amount of time, in seconds, for which the server will maintain a persistent connection with a client. The default value is 30 seconds.

--license <file-name>

The name of an existing edit file containing the LightWave Client product license. If this option is omitted, the license file is located according to Product Licensing rules.

--log [{ <destination> | * } [level [format]] | +<log-config-file>]

Specifies the process log location, the level, and the log event format, or the location of a log configuration file. The destination value may be a process name, a file name, or the asterisk (*) character. If the asterisk is used then the log output is directed to the home term of the process. The level value may be "error", "warning", "info", or "debug" and controls the type of information that is output to the log destination. The "error" level produces the least output while the "debug" level produces the most output. The format value may be "text" indicating that the log events should be output as text strings or "event" indicating that the log events should be output in EMS event format. If omitted, the default is "--log * info text". See Using Configuration Files for information about logging configuration files.

--max-tx-timeout <seconds>

Specifies the maximum transaction timeout value that client applications may specify in a request (using the lw-transaction-timeout header). If a client application specifies a larger value, max-tx-timeout will be used instead. If this option is omitted, the default value 0 is used, which indicates that the NonStop system Auto Abort timer will determine the transaction timeout.

--monitor <option>[:<interval>] [<option>[:<interval>]] ...

Enables file monitoring and specifies the monitoring interval. If the interval is omitted, the default value is 15 seconds. The following files may be monitored: log. See Using Configuration Files for information about monitoring log configuration files.

--monitor-workers <seconds>

Specifies the number of seconds that an SWORKER process must be running before the SERVER process will restart it in the event of an SWORKER abend. This prevents the SERVER process to from continuously restarting an SWORKER process that is abending on startup. The default value is 60 seconds. Specify 0 seconds to disable monitoring of SWORKER processes.

--msg-log <file-name>

Specifies the location of the Message Logging configuration file. See Configuring Message Logging for information about Message Logging configuration.

--service-ports <port specification> [<port-specification>] ...

Specifies a list of TCP/IP ports that the SERVER process should listen on for service (API) connections. <port-specification> has the following form:

- [<tcpip-process>:]<port-number>[:<certificate-name>] [:host=<host-name>][:bindaddr=<ip-address>][:filterkey=<filter-key]

where:

- <tcpip-process> - The NonStop TCP/IP process to use for this port. If omitted, the value of the define =TCPIP^PROCESS^NAME is used if it exists, otherwise $ZTC0.
- <port-number> - The TCP/IP port to listen on, 0-65535. Port numbers 0-1023 can only be used if run by a member of the SUPER group.
- <certificate-name> - The CN value of the X509 certificate to use for this port. If specified, the port will use HTTPS protocol.
- <host-name> - The host name to use for this port. This host name will be used by the Console API Tester and Swagger / OpenAPI download features. This option must be supplied with the service port is not using the same TCPIP provider as the console port.

- <ip-address> - The IP address to bind to when the TCP/IP provider has multiple IP addresses configured.
- <filter-key> - A file specification to be used to set the =PTCPIP^FILTER^KEY value for the TCP/IP listen socket, which allows multiple binds to the same IP and port by more than one application per processor

Multiple port specifications are separated by spaces. If this option is omitted, no service ports will be opened and therefore no services (APIs) will be available to client applications. See Server Certificates for information about installing server certificates.

--service-sts-max-age

Enables the Strict-Transport-Security header in Service responses and specifies the max-age value in seconds. If omitted, the header is not returned in Service responses.

--show-server-id

If present, causes LightWave to include a "Server" HTTP header in responses, otherwise it is omitted. The content of the "Server" header includes the product name, "LightWave Server", and the product version number. Although once customary for HTTP servers to include, the practice is now considered a security leak.

--shutdown <server-process-name> [!]

Initiates an orderly shutdown of a SERVER or SWORKER process running with the process name <server-process-name>. When given the name of a SERVER process, any associated SWORKER processes are also shut down. The optional ! argument requests a "quick" shutdown, which causes the processes to terminate without waiting for any pending requests to complete. If the --shutdown option is used, all other options are ignored.

--sts-max-age <seconds>

Enables the Strict-Transport-Security header in Console responses and specifies the max-age value in seconds. If omitted, the header is not returned in Console responses.

--tls-cipher-list <cipher-name-list> | +<cipher-list-file>

Specifies the list of ciphers to be used for TLS connections. The cipher list may be specified as a string containing a list of ciphers or an EDIT file containing a list of ciphers. Cipher names are specified using OpenSSL format. For more information on OpenSSL ciphers to OpenSSL Ciphers.

--tls-disable-v1.0

Disables TLS v1.0 connections to the console. If omitted, TLS v1.0 connections are allowed. ***Deprecated, use --tls-protocols.***

--tls-disable-v1.1

Disables TLS v1.1 connections to the console. If omitted, TLS v1.1 connections are allowed. ***Deprecated, use --tls-protocols.***

--tls-protocols   [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3]

Specifies the list of protocols that should be available for TLS connections. If omitted, the default value is "TLSv1.2 TLSv1.3".

Remarks

All command-line-option names and values are case-insensitive except where noted. If multiple occurrences of the same command line parameter are encountered, the setting of the last occurrence is used.

When the -log option format is set to event, EMS events will be sent to the output device with the following EMS Subsys ID:

Z-OWNER	"NUWAVE"
Z-NUMBER	3
Z-VERSION	Product major version

Examples

Start the SERVER process as a process pair in CPU 0 and 1. The Console may be accessed on port 8080 using HTTP or port 8443 using HTTPS. Services may be accessed on port 9080 using HTTP or 9443 using HTTPS. This example assumes that a server certificate with Common Name `lightwave.example.com` has already been installed. See Server Certificates for information about installing server certificates.

```
tacl> run server / name $LWS, nowait, cpu 0 / --backupcpu 1 &
--console-ports $ztc0:8080 $ztc0:8443:lightwave.example.com &
--service-ports $ztc0:9080 $ztc0:9443:lightwave.example.com &
--log $zhome info
```

Shut down LightWave Server and any attached SWORKER processes:

```
tacl> run server --shutdown $LWS
```

More Information

See Configuration Best Practices for additional information.

SUTIILITY

⚠ SUTILITY has been deprecated and will no longer be updated. LWSCOM has replaced SUTILITY as the LightWave Server management CLI.

The SUTILITY program is used to perform LightWave Server management tasks from the TACL command line. The general syntax the start the SUTILITY program is:

```
tacl> run SUTILITY / run-options / command-line-options
```

> **Important**
>
> Always run SUTILITY with the --update-filesystem option after installing a new version of LightWave Server.

run-options

The standard TACL run options. Note that process does not open the IN or OUT file.

command-line-options

@<command-file>

> Reads command line options from <command-file>. Options specified on the command line override any duplicates specified in the file. At most, one '@' option may used. The file itself cannot contain an '@' option (i.e., no nesting).

--clean

> When items are deleted from the filesystem via the Console, they are only marked for deletion and not immediately physically deleted. Rather, the data is physically deleted over time by a housekeeping process within SERVER. The –clean option immediately 'cleans up' (physically deletes) any data currently marked for deletion. Use this option

if you want to remove unused data from the filesystem without waiting for the housekeeping process to run.

--create-filesystem

Creates a new LightWave Server filesystem in the location specified by the --filesystem option.

--filesystem <subvolume-spec>

Specifies the location of the target LightWave Server filesystem. <subvolume-spec> may optionally include a volume name. A value of '*' indicates the current subvolume. The filesystem must be located on a TMF-audited volume. Refer to Managing the Filesystem for more information.

--set-password <username> <password>

Overwrites the password for the named user.

--update-filesystem

Updates a LightWave Server filesystem in the location specified by the --filesystem option with the latest installed product revisions.

SWORKER

SWORKER is the LightWave Server Worker process. When a client creates a TCP/IP connection to a LightWave Server service port, the SERVER process delegates the connection and all future processing of that connection to the SWORKER process having the least number of active connections.

Starting a Worker Process

A worker process is started by running the SWORKER program from TACL. Multiple worker processes can be attached to the same SERVER process.

```
tacl > run SWORKER / run-options / command-line-options
```

You should be logged-on as a user with sufficient privileges to access the system resources that the process requires.

run-options

The standard TACL run options. The CPU option is recommended in order to load-balance across multiple cpus. The NOWAIT options is recommended. The TERM options is also recommended if started from a dynamic terminal device. The IN and OUT options are ignored.

command-line-options

@<command-file>

> Reads command line options from <command-file>. Options specified on the command line override any duplicates specified in the file. At most, one '@' option may used. The file itself cannot contain an '@' option (i.e., no nesting).

--server <process-name>

> Specifies the name of the SERVER process that will delegate connections to this worker. This option is required. The SERVER process must be running on the same node as the SWORKER process.

Remarks

SWORKER uses the same logging options as the SERVER it is associated with.

Examples

Start the SERVER process as a process pair in CPU 0 and 1 with SWORKER processes in CPU 2 and 3.

```
tacl> run server / name $LWS, nowait, cpu 0 / --backupcpu 1 &
--console-ports $ztc0:8080 $ztc0:8443:lightwave.example.com &
--service-ports $ztc0:9080 $ztc0:9443:lightwave.example.com &
--log $zhome info
tacl> run sworker / name, nowait, cpu 2 / --server $LWS
tacl> run sworker / name, nowait, cpu 3 / --server $LWS
```

Event Message Reference

This section describes the LightWave Client process event messages. Event numbers are grouped into the following ranges:

Level	Range
Error	1000-1999
Warning	2000-2999
Information	3000-3999

Events messages use the following EMS Subsystem ID and event subject:

Process	Subsystem Owner	Subsystem Number	Version	Event Subject
SERVER	NUWAVE	3	1	SERVER
SWORKER	NUWAVE	3	1	SWORKER

- SERVER / SWORKER Process Events

SERVER / SWORKER Process Events

1100

ERROR object discarded event *event-type*.

Cause

An internal error occurred which caused an event to be discarded.

Effect

The effect is unknown.

Recovery

This error should be reported to the LightWave Server system administrator.

1110

ERROR *port-type* port *tcpip-process:port* bind failed, *reason*

Cause

An error occurred binding to the TCP/IP port due to the indicated *reason*.

Effect

The port cannot start.

Recovery

Correct the underlying condition if possible. An attempt will be made to automatically restart the port.

1111

ERROR *port-type* port *tcpip-process:port* accept failed, *reason*

Cause

An error occurred accepting a connection on the TCP/IP port due to the indicated *reason*.

Effect

The connection fails.

Recovery

Correct the underlying condition if possible.

1112

ERROR *port-type* port *tcpip-process:port* listen, *reason*

Cause

An error occurred binding to the TCP/IP port due to the indicated *reason*.

Effect

The port cannot start.

Recovery

Correct the underlying condition if possible. An attempt will be made to automatically restart the port.

1113

ERROR Start of *port-type* port *tcpip-process:port* failed, retry in 15 seconds

Cause

A port restart has been scheduled due to an error.

Effect

The port is stopped and will be restarted.

Recovery

None.

1114

ERROR IO error *filesystem-error* occurred on *port-type* port *tcpip-process:port*, the port will restart.

Cause

An I/O error occurred on the port.

Effect

The port will be restarted.

Recovery

Correct the underlying condition if possible. An attempt will be made to automatically restart the port.

1115

ERROR *port-type* port *tcpip-process:port* create socket failed, error *error-number: reason*

Cause

An error occurred while attempting to create a socket due to the indicated *reason*.

Effect

The port cannot start.

Recovery

Correct the underlying condition if possible. An attempt will be made to automatically restart the port.

1201

ERROR The filesystem in subvolume *subvolume-name* could not be opened, *reason*

Cause

The filesystem in the specified *subvolume-name* could not be opened for the supplied *reason*

Effect

The process cannot start.

Recovery

Correct the --filesystem startup parameter and restart the process

1202

ERROR Startup cannot continue.

Cause

An error occurred that prevents startup of the process.

Effect

The process stops.

Recovery

Correct the error and restart the process.

1203

ERROR Unable to start *port-type* port *port-specification*: *error-description*

Cause

An error occurred while trying to start a TCP/IP listen on the specified Console or Service port using the *port-specification*. The *error-description* describes the error

Effect

The process will attempt to restart the port in 15 seconds.

Recovery

If the error can be retried and the retry succeeds, no recovery action is necessary. If the retry fails due to a configuration error, correct the error and restart the process.

1204

ERROR Filesystem in *subvol* is already opened by SERVER process *process-name*. SERVER processes cannot share a filesystem.

Cause

A SERVER process was started using a filesystem in *subvol* that is already in use by another SERVER process *process-name*. Filesystems cannot be shared among multiple SERVER processes.

Effect

The process stops.

Recovery

Start the SERVER process using a filesystem that is not already in use.

1205

ERROR *license-error-description*

Cause

A general license error has occurred. The *license-error-description* describes the error.

Effect

The process cannot start.

Recovery

Correct the license error and restart the process.

1207

ERROR Invalid signature on message *message-number*

Cause

An SWORKER process received an inter-process message that had a valid *message-number*, but an invalid signature. This indicates that a process other than the SERVER process has sent a message to the SWORKER process.

Effect

The message is discarded.

Recovery

No recovery is required.

1208

ERROR Unrecognized receive msg *message-number*

Cause

An SWORKER process received an inter-process message with *message-number* that was not recognized. This indicates that a process other than the SERVER process has sent a message to the SWORKER process.

Effect

The message is discarded.

Recovery

No recovery is required.

1210

ERROR The supplied *default-tx-timeout* or *max-tx-timeout* option is invalid or inconsistent: reason

Cause

The supplied startup options for *default-tx-timeout* or *max-tx-timeout* are invalid or the default timeout is greater than the max timeout.

Effect

The process cannot start.

Recovery

Correct the option value(s) and restart the process.

1211

ERROR The --tls-options option value *value* is invalid. *reason*

Cause

The specified option *value* is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1212

ERROR The --tls-cipher-list option value *value* is invalid. *reason*

Cause

The specified option *value* is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1213

ERROR Worker *process-name* has abended and will be restarted.

Cause

The SWORKER process *process-name* abended.

Effect

The SWORKER process will be restarted using the same *process-name* and in the same CPU.

Recovery

This error should be reported to the LightWave Server system administrator.

1214

ERROR Worker *process-name* has abended *count* seconds after starting. It will not be restarted.

Cause

The SWORKER process *process-name* abended *count* seconds after starting, which is less that the threshold set with the –monitor-workers startup option.

Effect

The SWORKER process is not restarted, in order to avoid repeated ABENDs.

Recovery

This error should be reported to the LightWave Server system administrator.

1215

ERROR Unable to monitor worker *process-name*, *reason*

Cause

The SERVER process was unable to initiate monitoring of the SWORKER process *process-name* due to *reason*.

Effect

The SWORKER process is not monitored for abnormal termination.

Recovery

Determine the cause of the error based on *reason*. Restart the SWORKER process if necessary to resolve the issue.

1216

ERROR Worker *process-name* restart failed, *reason*

Cause

The SERVER process was unable to restart SWORKER process *process-name* after an abnormal termination, due to *reason*.

Effect

The SWORKER process is no longer running and will not be restarted.

Recovery

Determine the cause of the error based on *reason*. Restart the SWORKER process if necessary to resolve the issue.

1220

ERROR The --default-encoding option value *value* is invalid. The encoding is not available.

Cause

The specified option *value* is not a valid or available encoding.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1221

ERROR The --tls-protocols option value *value* is invalid. *reason*

Cause

The specified option *value* is invalid due to *reason*.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1222

ERROR The --monitor option value *value* is invalid. *reason*

Cause

The specified option *value* is invalid due to *reason*.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1223

ERROR The --blob-files option value *value* is invalid. *reason*

Cause

The specified option value *value* is invalid due to *reason*.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1230

ERROR Index error on filesystem *subvol*. *reason*

Cause

There is an error or inconsistency in the alternate index file settings for the filesystem in *subvol*. The error detail is contained in *reason*.

Effect

The process cannot start.

Recovery

Correct the issue with the filesystem alternate indexes and restart the process.

1231

ERROR Unable to purge BLOB file *file-name*, file system error *error-number* : *error-text*

Cause

A file system error occurred when trying to delete BLOB file *file-name*. The error is indicated in *error-number* and *error-text*.

Effect

The BLOB file was not deleted and remains on disk.

Recovery

Resolve the issue indicated by the *error-number* and *error-text*. Manually delete the BLOB file if appropriate.

1232

ERROR The --sts-max-age option value *value* is invalid. The value must be supplied as a positive number of seconds.

Cause

The option *value* was not specified as a positive number of seconds.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process

1233

ERROR The --service-sts-max-age option value *value* is invalid. The value must be supplied as a positive number of seconds.

Cause

The option *value* was not specified as a positive number of seconds.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process

1261

ERROR The specified license file *file-name* could not be opened, file system error: *error-description*.

Cause

The indicated file system error occurred while opening the license file.

Effect

The process cannot start.

Recovery

Correct the condition that caused the error and restart the process.

1262

ERROR The specified license file *file-name* is invalid.

Cause

The specified file does not contain a valid product license

Effect

The process cannot start.

Recovery

Install a valid product license file.

1270

ERROR The --msg-log option must have a value.

Cause

The --msg-log option was specified without a value.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1271

ERROR Message logging configuration is invalid. *reason*

Cause

The message logging configuration contains a syntax error.

Effect

The process cannot start or if already running and the configuration is monitored, the configuration cannot be reloaded.

Recovery

Correct the error and restart the process or if the configuration is monitored, wait for the configuration to reload.

1272

ERROR The message logging configuration file *file-name* could not be accessed. *reason*

Cause

The message logging configuration file could not be loaded due to the indicated *reason*.

Effect

The process cannot start or if already running and the configuration is monitored, the configuration cannot be reloaded.

Recovery

Correct the error and restart the process or if the configuration is monitored, wait for the configuration to reload.

1273

ERROR Message logging configuration is invalid. *reason*

Cause

The message logging configuration file could not be loaded due to the indicated *reason*.

Effect

The process cannot start or if already running and the configuration is monitored, the configuration cannot be reloaded.

Recovery

Correct the error and restart the process or if the configuration is monitored, wait for the configuration to reload.

1274

ERROR Message logging collector *collector-name*: IO retry limit exceeded. The collector has been disabled.

Cause

Excessive errors occurred while writing events to the specified collector.

Effect

No further events will be sent to the collector.

Recovery

Correct the underlying condition and restart the process.

1275

ERROR Message logging collector *collector-name*: *io-type* start IO error: *serverclass:pathmon-name:serverclass-name*, error *filesystem-error:pathsend-error*. The I/O will be retried in interval seconds.

Cause

The indicated error occurred while sending an event to the collector.

Effect

The event is queued for retry.

Recovery

The process will attempt to recover from the error.

1283

ERROR Error loading log configuration from file *file-name*. *reason*

Cause

The process log configuration file could not be loaded due to the indicated *reason*.

Effect

The process cannot start or if already running and the configuration is monitored, the configuration cannot be reloaded.

Recovery

Correct the error and restart the process or if the configuration is monitored, wait for the configuration to reload.

2201

WARN *backup-takeover-message*

Cause

The backup process has taken over for the primary process for the reason indicated.

Effect

The backup process is not the primary. A new backup process will be created.

Recovery

Informational message only; no corrective action is needed.

2202

WARN Backup CPU *cpu* is down.

Cause

The configured CPU for the backup process and the process itself are down.

Effect

There is no longer a backup process running.

Recovery

When the backup *cpu* is restored, a new backup process will be started automatically.

2203

WARN Unable to start backup process in CPU *cpu*, *reason*

Cause

The backup process could not be started for the indicated *reason*.

Effect

There is no longer a backup process running.

Recovery

If possible, correct the underlying condition. The primary process will attempt to restart the backup periodically.

2204

WARN Backup CPU *cpu* is up.

Cause

This event is used for informational purposes.

Effect

A new backup process will be started.

Recovery

Informational message only; no corrective action is needed.

2205

WARN Unable to open worker *process-name* for configuration.

Cause

The SWORKER process *process-name* sent a registration request to the SERVER process, but the SERVER process could not open the SWORKER process. This is likely due to a security error.

Effect

The SWORKER registration fails.

Recovery

Determine the cause of the open failure and restart the SWORKER process.

2206

WARN Certificate *cert-name* for *port-type* port *port-name* does not exist. Connections will be rejected until the certificate is installed.

Cause

An HTTPS port was configured but the specified certificate is not installed.

Effect

TLS connections will fail until the certificate is installed.

Recovery

Informational message only; no corrective action is needed.

2207

WARN SWORKER processes are over capacity, connections are being delegated to the SERVER process.

Cause

The SERVER process cannot delegate an incoming connection to an SWORKER process, indicating that the number of SWORKER processes configured is not sufficient to handle the application load.

Effect

The SERVER process accepts and handles the connection.

Recovery

Start additional SWORKER processes sufficient to handle the application load.

2208

WARN Unable to load ICU data file *reason*. Character encoding is limited to ISO-8859-1

Cause

An error occurred while attempting to load the ICU data file for the indicated *reason*.

Effect

ICU will not be used for character encoding conversions, which limits character encoding to ISO-8859-1.

Recovery

If character encodings other than ISO-8859-1 are required, correct the cause of the error and restart the process.

2209

WARN ICU is disabled. Character encoding is limited to ISO-8859-1.

Cause

The --disable-icu startup option was supplied, disabling ICU character encoding support.

Effect

ICU will not be used for character encoding conversions, which limits character encoding to ISO-8859-1.

Recovery

If character encodings other than ISO-8859-1 are required, restart the process without the --disable-icu startup option.

2210

WARN The filesystem is *percent-value* full. Perform filesystem maintenance as soon as possible.

Cause

The filesystem is nearly full.

Effect

None.

Recovery

Perform filesystem maintenance. See Managing the Filesystem

2211

WARN The filesystem is *percent-full* full. HTTP logging has been automatically disabled.

Cause

The filesystem is nearly full and HTTP logging has been disabled.

Effect

HTTP logging has been disabled to avoid filling the filesystem.

Recovery

Perform filesystem maintenance. See Managing the Filesystem

2212

WARN The filesystem is *percent-full* full. Diagnostic logging has been automatically disabled.

Cause

The filesystem is nearly full and diagnostic logging has been disabled.

Effect

Diagnostic logging has been disabled to avoid filling the filesystem.

Recovery

Perform filesystem maintenance. See Managing the Filesystem

2230

WARN Index warning for filesystem *filesystem-subvol*. *reason*

Cause

There may be an inconsistency with the Enscribe alternate indexes references for the filesystem located in *filesystem-subvol*. Additional details are provided in *reason*.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3101

INFO Started *port-type* port *port-specification*

Cause

Indicates that a Console or Service port was successfully started using the *port-specification*.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3201

INFO *component-name - component-version*

Cause

This event shows the product component name and version.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3202

INFO Running in CPU *cpu*.

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3203

INFO Backup process started in CPU *cpu*

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3204

INFO Stopped.

Cause

The process has stopped.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3205

INFO Opened filesystem in *filesystem-subvol*.

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3206

INFO Server process *process-name* has stopped.

Cause

The SERVER process indicated by *process-name* has stopped normally. This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3207

INFO Loading product license from *file-name*

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3208

INFO Registered Worker *process-name*

Cause

Indicates that the SWORKER process with *process-name* has registered with the SERVER process.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3209

INFO Deregistered Worker *process-name*

Cause

Indicates that the previously registered SWORKER process with *process-name* is no longer registered with the SERVER process.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3210

INFO Transaction default timeout *default-timeout*, max timeout *max-timeout*.

Cause

Indicates that the TMF transaction default timeout is *default-timeout* seconds, and the maximum TMF transaction timeout is *max-timeout* seconds. For more information, see the *--default-tx-timeout* and *--max-tx-timeout* options in the SERVER Command Line Reference.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3211

INFO Using TLS options: *options*

Cause

Indicates that the TLS options specified by the --tls-options startup option were successfully loaded.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3212

INFO TLS cipher list: *cipher-list*

Cause

Indicates that the TLS cipher list specified by the --tls-cipher-list startup option was successfully loaded.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3213

INFO Worker monitoring is enabled | disabled. *minimum-up-time*

Cause

Indicates that SWORKER process monitoring by the SERVER process is enabled or disabled. If enabled, the *minimum-up-time* value is displayed in seconds. For more information, see the *--monitor-workers* option in the SERVER Command Line Reference.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3214

INFO Worker *process-name* has stopped.

Cause

Indicates that the SERVER process has received a system STOP message for SWORKER process *process-name*.

Effect

The SWORKER process is no longer running and will be deregistered.

Recovery

Informational message only; no corrective action is needed.

3215

INFO Worker *process-name* restarted in CPU *cpu*.

Cause

Indicates that the SERVER process has restarted SWORKER process *process-name* in CPU *cpu*, after the SWORKER process stopped abnormally.

Effect

The SWORKER process is restarted.

Recovery

Informational message only; no corrective action is needed.

3218

INFO Using default encoding *encoding-name*

Cause

Indicates that the default encoding to use for character string conversions for all services is *encoding-name*. For more information, see the *--default-encoding* option in the SERVER Command Line Reference.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3221

INFO TLS protocols available: *protocol-list*

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3222

INFO Configuration monitoring is disabled.

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3223

INFO Configuration monitoring is enabled: *monitor-config*

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3225

INFO BLOB file config is 'pattern=*pattern* user=*user* security=*security-string* extents=*extents*'.

Cause

Indicates that BLOB files will be created using the *pattern*, *user*, *security-string*, and *extents* indicated. For more information, see the --*blob-files* option in the SERVER Command Line Reference.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3226

INFO Loaded process log configuration from file *file-name*.

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3227

INFO Process log configuration is *process-log-config*

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3228

INFO Server process has switched. Renewing registration.

Cause

This SWORKER process has detected that the SERVER process has switched from primary to backup.

Effect

The SWORKER process will renew its registration with the new primary SERVER process.

Recovery

Informational message only; no corrective action is needed.

3229

INFO Console HSTS max-age is *count* seconds.

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3230

INFO Service HSTS max-age is *count* seconds.

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3240

INFO Shutdown started.

Cause

Process shutdown has started.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3270

INFO Loading message logging configuration from *file-name*.

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

Release Notes

The Release Notes contain important information about the software. Review all sections before installing or upgrading the product.

Installation Prerequisites

LightWave Server requires the following HP NonStop Server software:

- One of the following NonStop System RVUs:
  - TNS/E: H06.09 or later, J06.03 or later.
  - TNS/X: L15.08 or later.
- HP NonStop TCP/IP
- HP NonStop Transaction Management Facility (TMF)

Although not required, LightWave Server integrates with these products, if installed:

- HP NonStop TS/MP (Pathway). Large message support is available when supported by TS/MP.
- HP Data Definition Language (DDL or DDL2).
- HP TCP/IP Parallel Library, or TCP/IPv6. LightWave Server currently supports IPv4 mode only.

The Console supports the following browsers with versions N through N-1, where N is the current browser version:

- Chrome
- Edge
- Firefox
- Safari
- Internet Explorer

Upgrade Considerations

- When upgrading from a previous release the filesystem update procedure must be performed. See Updating LightWave Server for more information.
- The SUTILITY program has been deprecated. All CLI based product management should be done using LWSCOM.
- The installation and upgrade procedures have been updated to use LWSCOM.

Product Licensing

LightWave Server requires the installation of a license. The license is stored in an edit file on the NonStop Server that is read by the product in order to validate the installation. A license is a multi-line text block similar to that shown below:

```
-----BEGIN LICENSE-----
product=lightwave_server
systemNumber=012345
expiration=6/1/2016
supportExpiration=6/1/2016
restrictedLicense=no
signature=172
T2M23bpqxRzyEgt2cETwRb8j5DMkGdVzyM2WZTRsssoIZXooc6
ROAtJdOevToYeX10/UPCSVZJynUZO/uf7MSxVn2FnA5LH1g87g
3F9wDvajQNdiRxrX6rHThNovxH+dKOXKb+nzkGZx47PXF4izA1
W4GJhmAJi9n7z6x5WZEyE=
-----END LICENSE-----
```

When the license is received from NuWave Technologies it must be copied to an edit file on the NonStop System *exactly* as provided. The product programs search for the license file in the following locations and in the following order:

- The file "LICENSE" in the LightWave SERVER program subvolume
- The file "LICLWS" in the LightWave SERVER program subvolume
- The file "$SYSTEM.NUWAVE.LICLWS"

License Expiration

The license contains two expiration date fields to be aware of:

expiration

This field contains the date that the license will expire, or "none" if the license never expires. For licenses that expire, once the expiration date has passed the software will continue to function but will log license violation errors to the process log. For licenses that don't expire, the software will run indefinitely.

supportExpiration

This field contains the date that your current software support agreement expires. Once this expiration date has passed the software will continue to run with full functionality (subject to the *expiration* date) on your current release but you may not upgrade the software to a version that was released after the *supportExpiration* date. Software with a release date after the *supportExpiration* date will shutdown immediately with a license error. The release date for a particular software version can be found in the Upgrade Guide or the product VPROC.

When you renew your support agreement we will email you a new license. You do not need to install the new license until the next software upgrade.

Cumulative Change Log

- 1.0 Releases
- 1.1 Releases
- 1.2 Releases
- 1.3 Releases

1.0 Releases

1.0.11 - 27 Mar 2021

Problems Corrected

[LS-859] - LWSCOM INFO API command may report incorrect number of APIs
[LS-879] - Chunked encoded request returns HTTP 400 status when the chunk & chunk trailer are not received in the same socket recv buffer
[LS-898] - LWSCOM pagesize command may report valid input as invalid
[LS-905] - Process logger may leak memory when debug logging is enabled
[LS-912] - Setting isSet="1" does not work for JSON object serialization when the object is empty
[LS-914] - API tester no longer works in Chromium 87+ based browsers
[LS-917] - API Tester will not launch in IE11

1.0.10 - 27 Aug 2020

Problems Corrected

[LS-437] - Add Intermediate Certificate silently ignores bad PEM input
[LS-571] - LWSCOM prints incomplete error message when attempting to export the test service
[LS-649] - Dictionary import fails when LightWaveAttribute is invalid.
[LS-679] - Swagger download returns internal server error if the dictionary associated with an API is not installed.
[LS-751] - SERVER does not detect when filesystem is not TMF audited, resulting in unpredictable behavior
[LS-753] - HTTP log file name sequence does not roll over when date changes
[LS-757] - HTTP log version & date directives have incorrect values
[LS-758] - Console HTTP log time ranges should be in GMT to match the log timezone
[LS-790] - Fix typo in LWSCOM HTTPLOG help text
[LS-798] - A structure element in an array of structures is not serialized if the element is hidden in a previous structure in the array
[LS-805] - Diagnostic log is corrupted if line break occurs at multi-byte character
[LS-808] - HSTS header may be sent intermittently
[LS-816] - Auto begin transaction is suspended instead of aborted when request results in I/O or undefined reply error
[LS-842] - unsigned long long values > 9223372036854775807 are not de/serialized correctly

New Features

[LS-807] - Implement XSRF protection in the Console

1.0.9 - 04 MAR 2020

Problems Corrected

[LS-617] - Dictionary import sets arbitrary limit of 100 objects
[LS-259] - Documentation is missing one or more command line option descriptions
[LS-620] - Dictionary refresh no longer pre-selects existing DDL objects for refresh
[LS-631] - Certificate installation dialogs with textarea fields have no Cancel / Install buttons
[LS-650] - API tester host:port should default to port with the same scheme as the browser
[LS-671] - SERVER/SWORKER abends when operation process name is mapped from a parameter and the parameter is not supplied with the request
[LS-684] - LWSCOM may exit with error -3: Process called STOP or ABEND with erroneous arguments
[LS-686] - Filesystem deadlock may occur during service redeploy
[LS-693] - API editor does not refresh when a new version of an API is uploaded
[LS-694] - LWSCOM: service name is not printed when import specifies * for name

[LS-697] - Certificate import with duplicate CN does not replace existing certificate
[LS-698] - LWSCOM import cert does not replace existing cert with same common name
[LS-710] - Transaction and dialog APIs are not longer accessible from the Console
[LS-717] - Organization field missing in generated CSR
[LS-727] - Upgrade AngularJS to version 1.7.9 to mitigate security vulnerabilities
[LS-728] - Console uses mixed content links in footer
[LS-733] - --http-keepalive-timeout & --http-keepalive-max startup options are not processed correctly

New Features

[LS-712] - Upgrade to OpenSSL 1.0.2t
[LS-695] - Add option to disable digest authentication
[LS-731] - Allow selection of diagnostic log content
[LS-732] - Add isSet schema attribute to indicate presence of optional fields. See Working with Optional Elements.
[LS-736] - Allow substitution variables to be supplied as constant values. See Working with User Defined APIs.
[LS-749] - Add hideIfEmpty schema attribute to omit serialization of empty elements. See Working with Optional Elements.

1.0.8 - 05 MAR 2019

Problems Corrected

[LS-438] - Swagger generator does not follow CORS policy of associated service's access control policy
[LS-466] - Diagnostic log view hangs when log contains invalid content
[LS-483] - Swagger generator leaks memory
[LS-552] - Service installation silently fails when the filesystem is full.
[LS-563] - SERVER/SWORKER can hang when processing a chunked transfer with very large chunk size
[LS-566] - Console may log out due to inactivity while using the API editor for a long period without saving
[LS-567] - API I/O options does not allow 5 character process names when qualified with node name
[LS-569] - Swagger download returns invalid ETAG
[LS-576] - Authorization header is not masked in the diagnostic log
[LS-578] - Diagnostic log settings shows logging enabled with 0 minutes remaining
[LS-579] - Diagnostic log delete buttons are enabled when there are no logs
[LS-581] - Command line consisting only of a comma is reported as command abbreviation
[LS-577] - Sensitive data masking diagnostic log warning is not displayed consistently

New Features

[LS-586] - Add options to disable TLS v1.0 & v1.1
[LS-587] - Console returns X-Content-Type-Options, X-Frame-Options, X-XSS-Protection, Strict-Transport-Security security headers.

1.0.7 - 05 OCT 2018

Problems Corrected

[LS-453] - On deserialization, actual array count overrides count specified in dependsOn element.
[LS-457] - HTTP and/or Diagnostic logs can become corrupted with many SWORKERS under high volume
[LS-461] - Connection delegation to SWORKER can fail under high load
[LS-468] - SERVER/SWORKER memory leaks
[LS-469] - Deadlock occurs when SWORKER tries to refresh APIs while SERVER is redeploying.
[LS-475] - SERVER/SWORKER abends if logging is enabled but no services are selected.
[LS-479] - Memory performance tuning
[LS-489] - LWSCOM "delete diaglog' with wildcard does not work as documented.
[LS-490] - LWSCOM "control diaglog" command inconsistent with Console UI.
[LS-493] - LWSCOM exported diagnostic logs are written with incorrect line terminator
[LS-493] - EXPORTed diagnostic logs are written with incorrect line terminator
[LS-499] - Filesystem race condition may corrupt http/diagnostic log settings
[LS-500] - SERVER/SWORKER abends when digest auth header is sent with no qop parameter
[LS-503] - HTTP log time-taken fields is reported in microseconds, should be milliseconds.

New Features

[LS-339] - Add support for international character set encoding.
[LS-369] - Add support for sensitive data masking.
[LS-477] - Add custom Measure counters to SERVER/SWORKER
[LS-479] - Memory and performance tuning
[LS-502] - Disable all logging when filesystem exceeds 90% full.
[LW-748] - Add LWSCOM CLI utility

1.0.6 - 11 APR 2017

Problems Corrected

[LS-277] - passing alpha value as URI parameter mapped to integer field causes abend
[LS-322] - SERVER/SWORKER abend on when deserialization error occurs on param or header field.
[LS-329] - SERVER process may abend when initializing a field from the value property and the value is invalid for the type.
[LS-392] - Intermittent SERVER abend on shutdown

[LS-396] - Import Dictionary imports arbitrary xml files; does not allow deletion
[LS-406] - API Editor cannot map a field that contains '-'
[LS-408] - Array dependsOn field is not processed correctly when it references a field that is not an element of the array's parent structure
[LS-409] - Diagnostic log HTTP response shows request data when the response content type is octet/stream
[LS-410] - Console times out unexpectedly
[LS-414] - Swagger generation emits "integer" when source DDL is PIC 99V99.
[LS-434] - Errors for reply message reply code and HTTP status don't appear in IE 11.
[LS-442] - SERVER abends when malformed console/service-ports option is specified (TNS/E)
[LS-444] - 501 NOT IMPLEMENTED is returned for 'method not available' on existing path, should be 405 METHOD NOT ALLOWED

New Features

[LS-447] - Upgrade to OpenSSL 1.0.2o

1.0.5 - 14 NOV 2017

Problems Corrected

[LS-176] - Swagger link always points to test service interface
[LS-298] - Diagnostic shows the request body as empty if the request body cannot be parsed.
[LS-299] - Digest authentication fails if the user name is mixed case.
[LS-300] - stringPadding set at the API level is not honored
[LS-304] - Type object transfers strings to/from the message payload in UTF-8
[LS-311] - Deleted users & groups are not removed from access control policies
[LS-315] - Native API modules are registered and active even when --enable-native-apis option is not specified
[LC-324] - Float/double mapping not working
[LC-340] - Strings are transferred to/from the message payload in UTF-8
[LC-377] - sizeIs not working for types base64 and hexBinary
[LS-317] - Swagger generation emits all property types as "string"
[LS-318] Dictionary refresh button is enabled for dictionaries that are not imported from DDL

New Features

None

1.0.4 - 11 AUG 2017

Problems Corrected

[LS-152] - SERVER/SWORKER: CPU # in startup message is always 0.
[LS-158] - Console: Server certificate renewal completion toast shows shows "certificate deleted" instead of "CSR created".
[LS-159] - Console: unable to create user when name contains '-'.
[LS-163] - Memory leak in SWORKER process
[LS-186] - API store does not update the modification time when an API is saved
[LS-195] - API Editor: Editor shows warning icon for operation but reason for warning is nowhere to be found.
[LS-207] - Add API button doesn't work in FireFox
[LS-213] - API Editor: creating operation path throws invalid regex error.
[LS-214] - SERVER reports filesystem in use error if file FS02 is missing.
[LS-219] - Definitions containing certain types (e.g. SQL DATETIME) are not handled gracefully.
[LS-220] - Dictionary import erroneously assumes all fields named FILLER are character fields
[LS-221] - Nested Arrays with dependsOn are not serialized correctly
[LS-237] - http-keepalive-timeout and http-keepalive-max defaults are not handled correctly for console connections.
[LS-240] - Server stops listening on TCPIP port if the TCP/IP process restarts.
[LS-246] - Port file system error message is incorrectly formatted.
[LS-251] - SWORKER ignores --max-tx-timeout and --default-tx-timeout options.
[LS-252] - SERVER process shutdown behavior is inconsistent
[LS-262] - SUTILITY --set-password abends if password omitted
[LS-265] - SWORKER silently exits if --server not given a process name, or process name is not SERVER
[LS-268] - API tester: request and reply tabs are overlayed
[LS-269] - Fields following DDL REDEFINES are deserialized at the wrong offset.
[LS-271] - Serializer abends when first field is filled with padding character
[LS-280] - Creation of large diagnostic log silently fails.
[LS-286] - API Editor: Message Field selector returns incorrect field name when an array is selected.
[LS-292] - Certificate install fails when CSR common name and certificate common name don't match.
[LS-293] - Decimal values are serialized in scientific notation when the integer part of the value is 0.
[LS-241] - Dictionary importer generates filler elements without hide attribute.

New Features

[LS-155] - Add support for 2MB interprocess I/O.

[LS-231] - Support ISO-8601 to 64-bit timestamp de/serialization

[LS-238] - Automatically restart abended SWORKER processes

[LS-247] - Allow masking of sensitive data.

[LS-253] - Add support for –license command line option

[LS-273] - Add string padding indicator to binary message display in dictionary view

[LS-276] - Add IPM compression to diagnostic dumps.

[LS-289] - Add startup options to log.

[LS-229] - Add SSL session info to diagnostic log

[LS-244] - Add request processing time to diagnostic logs

[LS-218] - Allow filtering of DDL definitions at import; limit to import count

[LS-250] - Add a sample startup file for server

[LS-275] - Change default license file name to LICLWS.

1.0.3 - 03 NOV 2016

Problems Corrected

[LW-698] - Process abends if socket error occurs during accept phase.

New Features

None

1.0.2 - 28 OCT 2016

Problems Corrected

[LW-697] - SERVER/SWORKER process abends during shutdown

[LS-145] - Console: API method detail shows 'pattern attribute value: invalid escape'.

[LS-146] - Console: API 'user documentation' for API with no methods fails

[LS-149] - SERVER/SWORKER leaks memory on each request.

[LS-150] - Connections delegated to SWORKER may hang after first HTTPS connection

New Features

None

1.0.1 - 10 OCT 2016

This is the first GA release of LightWave Server.

1.1 Releases

1.1.9 - 15 July 2025

Problems Corrected

- Corrected BLOB response file handling when file cannot be opened.
- Corrected 400 response when request has `Expect: 100-continue` header and a request body.

Improvements

- Update to OpenSSL 3.5.1 to support quantum cryptography and security vulnerabilities.
- Update Libxml2 to version 2.14.4 to address security vulnerabilities.
- Added binary request/response body output to diagnostic log.

1.1.8 - 24 Sep 2024

Problems Corrected

- ERROR event 1216 is logged at INFO level.
- ERROR event 1202 is logged at INFO level.
- SWORKER process intermittently fails to register with error 201, when the SERVER process primary is abended.
- Update to OpenSSL 3.1.7 to mitigate security vulnerabilities.

Improvements

- Added support for binding console and service ports to a specific IP address when the TCP/IP provider is configured with multiple addresses. See the SERVER --console-ports and --service-ports options.

1.1.7.1 - 05 Apr 2024

Problems Corrected

- Dictionary type view JSON & BINARY tabs show no content.

1.1.7 - 24 Mar 2024

Problems Corrected

- Certificate chain building may fail if a self-signed certificate is installed.

- PKCS12 import fails if the file was encrypted with legacy ciphers.
- Change event number for message *INFO Shutdown started* from 1202 to 3240.
- The service-sts-max-age-option is not recognized on startup.
- Properties in JSON response are not serialized in the order defined in the API definition mappings.
- Update console footer documentation link.
- Update libxml2 to version 2.12.5.
- Update openssl to version 3.1.5.

1.1.6 - 27 Sep 2023

Problems Corrected

- --tls-cipher-list option does not accept TLS 1.3 specific cipher suites.
- Update OpenSSL to version 3.1.2 to mitigate security vulnerabilities.
- Update libxml2 to version 2.10.4 to mitigate security vulnerabilities.
- Swagger download does not work when service-ports and console-ports are not configured on the same TCPIP provide.

Improvements

- Added host name specification to the --service-ports startup option.
- Added the --service-sts-max-age option which allows configuring the Strict-Transport-Security HTTP header on service ports.
- All console download links now send session tokens as HTTP headers instead of query params.

1.1.5 - 27 Mar 2023

Problems Corrected

- Console exception is raised when an ACP is copied and then saved.
- Create dictionary shows an error when the subvolume node is not preceded with a backslash.
- Response properties are serialized as empty objects, when they are mapped individually in the response definition, from fields that are empty and have the hideIfEmpty attribute set.
- The API tester cannot be used, when the console ports and service ports use different TCPIP providers.
- SERVER/SWORKER may abend when message logging is configured and certain errors are returned from the collector serverclass.
- Sensitive data is not masked, when the field marked as sensitive in an element in a structure, and the structure is an item in an array.
- Update OpenSSL to version 1.1.1t to mitigate security vulnerabilities.
- Update Apache APR to version 1.7.2 to mitigate security vulnerabilities.

- Update APR-UTIL to version 1.6.3 to mitigate security vulnerabilities.

New Features

- API tester now allows entering an arbitrary base URL,

1.1.4 - 28 Sep 2022

Problems Corrected

- LightwaveAttributes in DDL comments may be ignored when embedded in a multi-line DDL comment.

Improvements

- Update PCRE version to mitigate vulnerabilities
- Add LWSCOM feature to validate licenses
- Add find feature (CTRL-F) to the API Tester

1.1.3 - 17 May 2022

Problems Corrected

- Multiple query param, header, & element mappings with the same name are deserialized incorrectly
- Request with a BLOB response may return an API compiler error
- Request returns 404 when a path component value contains the '.' or '*' character
- API Tester displays "extra" fields when multiple same name parameters are mapped to a single field
- HTTP & Diagnostic logging show as disabled on Console Dashboard when enabled (TNS/E only)
- SERVER / SWORKER may loop when an application server returns an IPM shorter than the dictionary definition
- SERVER may abend if there isn't a filesystem in the default subvol and the --filesystem startup option is not supplied
- LWSCOM environment is invalid if the TACL _DEFAULTS define contains the system name
- Dictionary import/refresh fails if DDL comments cannot be converted to valid XML comments

Improvements

- Performance improvements in JSON de/serialization and internal memory management
- API tester now uses an improved JSON editor
- Update to OpenSSL 1.1.1n to mitigate CVE-2022-0778
- Update libxml2 to v2.9.14 to mitigate security vulnerabilities.

1.1.2 - 28 Sep 2021

Problems Corrected

- Messaging logging reports non-0 CONNECT_HS_TIME when CONNECT_TIME is 0
- API editor does not allow isArray, nillable, or isNull property on element
- Response is serialized incorrectly if multiple Response BODY definitions are defined
- BLOB request returns HTTP 415 when content-type isn't application/json, octet-stream, x-www-form-urlencoded, or text/plain
- Issue with dictionary caching causes excessive filesystem I/O
- Filesystem housekeeping process uses unnecessary read lock operations
- Setting isNull property to non-zero does not work on structures
- Multiple query param, header, & element mappings with the same name are deserialized incorrectly
- Request with BLOB response may return an an API compiler error

Improvements

- Update to OpenSSL 1.1.1l

1.1.1 - 27 Mar 2021

Problems Corrected

[LS-914] - API tester no longer works in Chromium 87+ based browsers
[LS-917] - API Tester will not launch in IE11
[LS-922] - Message Logging meta data is only included when content spec includes "all"

Improvements

[LS-921] - Message logging sets IPM & HTTP lengths on all events, regardless of content selection
Update to OpenSSL 1.1.1k

1.1.0 - 25 Jan 2021

Problems Corrected (since v1.0.10)

[LS-859] - LWSCOM INFO API command may report incorrect number of APIs
[LS-879] - Chunked encoded request returns HTTP 400 status when the chunk & chunk trailer are not received in the same socket recv buffer
[LS-898] - LWSCOM pagesize command may report valid input as invalid
[LS-905] - Process logger may leak memory when debug logging is enabled
[LS-912] - Setting isSet="1" does not work for JSON object serialization when the object is empty

New Features

Add support for TLS 1.3
Add support for Message Logging
Add support for BLOBs
Allow LWSCOM users with read access to the filesystem to execute commands that don't modify the filesystem. See the ALLOW-READ-ACCESS option under the LWSCOM CONTROL FILESYSTEM command.
SERVER process now supports process log configuration using configuration files. See Using Configuration Files.
Add additional Measure counters lw-diag-log, lw-msg-log, lw-pathsend-max, lw-process-max. See Using Measure Counters.

1.2 Releases

1.2.1 - 15 July 2025

Problems Corrected

- Applies correct import options based on dictionary type during refresh from DDL.
- Added SOAPam Compatibility character substitution when UTF-8 to ISO-8859-1 conversion errors occur.
- Swagger `basePath` attribute corrected for a service with a base path defined.
- Corrected BLOB response file handling when file cannot be opened.

Improvements

- Update to OpenSSL 3.5.1 to support quantum cryptography and security vulnerabilities.
- Update Libxml2 to version 2.14.4 to address security vulnerabilities.
- Added binary request/response body output to diagnostic log.

1.2.0 - 27 Feb 2025

Problems Corrected

- Digest authentication no longer requires the opaque parameter.
- Corrected HTTP status 400 response when request has `Expect: 100-continue` header and a request body.
- Corrected nested XML array deserialization.
- Corrected issues if blob response file cannot be opened.
- Define propagation from SERVER to SWORKER now works as expected.
- Corrected an issue with field initialization when the application server reply IPM length is shorter than the method response type.

Improvements

- Added support for sharing filesystems among SERVER processes

- Added support for round-robin filter key (port sharing) for service ports.
- Added support for SOAPam Server Compatibility.

1.3 Releases

1.3.0 - 26 January 2026

Problems Corrected

- Fixed support for TLSv1 and TLSv2 protocols with OpenSSL 3.5.1.
- Corrected name conversion on SOAPam Compatibility DDL Refresh.
- Fixed port startup option to allow wildcarded domain names.

Improvements

- Added Auth Exits feature.
- Added Password Policy feature.

Third Party Software Licenses

LightWave Server incorporates code from several open source projects. The licenses for these projects are included below:

- AngularJS
- Angular Material
- Apache Portable Runtime
- Apache Portable Runtime Utility Library
- Apache HTTP Server
- dlmalloc Memory allocator
- International Components for Unicode
- Jansson JSON Toolkit
- cURL
- libxml2 XML Toolkit
- OpenSSL Toolkit
- Perl Compatible Regular Expressions

AngularJS

https://github.com/angular/angular.js/blob/master/LICENSE

Angular Material

Apache Portable Runtime

This product includes software developed by The Apache Software Foundation (http://www.apache.org/).

Portions of this software were developed at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign.

This software contains code derived from the RSA Data Security Inc. MD5 Message-Digest Algorithm.

This software contains code derived from UNIX V7, Copyright(C) Caldera International Inc.

http://apr.apache.org

http://www.apache.org/licenses/LICENSE-2.0

Apache Portable Runtime Utility Library

This product includes software developed by The Apache Software Foundation (http://www.apache.org/).

Portions of this software were developed at the National Center

for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign.

This software contains code derived from the RSA Data Security Inc.

MD5 Message-Digest Algorithm, including various modifications by Spyglass Inc., Carnegie Mellon University, and Bell Communications Research, Inc (Bellcore).

http://apr.apache.org

http://www.apache.org/licenses/LICENSE-2.0

Apache HTTP Server

This product includes software developed at The Apache Software Foundation (http://www.apache.org/).

Portions of this software were developed at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign.

This software contains code derived from the RSA Data Security Inc. MD5 Message-Digest Algorithm, including various modifications by Spyglass Inc., Carnegie Mellon University, and Bell Communications Research, Inc (Bellcore).

http://httpd.apache.org

http://www.apache.org/licenses/LICENSE-2.0

dlmalloc Memory allocator

Written by Doug Lea and released to the public domain.

http://gee.cs.oswego.edu/dl/html/malloc.html

http://creativecommons.org/publicdomain/zero/1.0/

International Components for Unicode

Copyright (c) 1995-2014 International Business Machines Corporation and others.

http://site.icu-project.org/

http://source.icu-project.org/repos/icu/icu/trunk/license.html

Jansson JSON Toolkit

Copyright (c) 2009-2014 Petri Lehtinen <petri@digip.org>

http://www.digip.org/jansson/

https://github.com/akheron/jansson/blob/master/LICENSE

cURL

Copyright (c) 1996 - 2014, Daniel Stenberg, <daniel@haxx.se>.

http://curl.haxx.se/libcurl/

http://curl.haxx.se/docs/copyright.html

libxml2 XML Toolkit

Copyright (C) 1998-2012 Daniel Veillard. All Rights Reserved.

http://www.xmlsoft.org/

http://opensource.org/licenses/mit-license.html

OpenSSL Toolkit

http://www.openssl.org

http://www.openssl.org/source/license.html

Perl Compatible Regular Expressions

http://www.pcre.org/

http://www.pcre.org/licence.txt

Appendices

Appendix A: The SOAPam Server Compatibility Feature

This Appendix describes the SOAPam Server Compatibility feature in LightWave Server.

Note: The SOAPam Server Compatibility feature is licensed separately from the base LightWave Server license. To acquire a license for this feature, please contact your NuWave account representative.

Introduction

The SOAPam Server compatibility feature allows SOAPam Server customers to migrate services from SOAPam Server to LightWave Server. Migration requires a few simple configuration and migration steps described below. The migration does not require modification to the SOAP client application or the existing NonStop applications.

How it Works

SOAPam Server services are migrated to LightWave Server by following these steps:

- Where appropriate, recreate SOAPam Server Users and Groups in LightWave Server.
- Create LightWave Server access control policies that will be applied to the migrated services.
- Using the LWSCOM utility, import the SOAPam Server Service Definition File (SDF) into the LightWave Server filesystem. The import process results in a LightWave Server API definition and accompanying Dictionary.
- Deploy the converted API as a LightWave Server Service. Client applications can then be configured to access the migrated service at the appropriate host address and port configured in LightWave Server.

For detailed instructions on migrating SOAPam Server services to LightWave Server, see Migrating SOAPam Server services to LightWave Server.

Migrating SOAPam Server services to LightWave Server

Create SOAPam Server Users and Groups in LightWave Server

Users and Groups in LightWave Server are conceptually similar to User and Groups in SOAPam Server. To migrate the Users and Groups, manually recreated any necessary SOAPam Server Users and Groups in LightWave Server. If your existing SOAPam Server services use Anonymous access, it may not be necessary to migrate any Users or Groups.

Create LightWave Server Access Control Policies

LightWave Server service access control is fundamentally different than SOAPam Server. The differences are summarized in this table:

SOAPam Server	LightWave Server
Access to services is controlled by setting access permissions on the File System folder containing the service SDF, using the File System Browser. The same access is granted for all services (SDFs) in the File System folder.	Access to services is controlled by creating Access Control Policies (ACPs) that can then be applied to deployed Services. Access Control Policies can be applied to one or more services.
User and Group access to a service is controlled by the User or Groups access to the File System folder containing the SDF.	User and Group access is specified in the Access Control Policy.
Service IP address restrictions are set at the Server level using Access Control Management.	IP address restrictions are set in the Access Control Policy.

Import the SOAPam Server Service Definition File into LightWave Server

The LWSCOM utility is used to import SDF files into LightWave Server, creating an API Definition and associated Dictionary. Throughout these examples, the SOAPam Server *echoStringSample* service. This service is contained in the SDF file *echoString.sdf* and is automatically installed in the SOAPam Server in folder */services/samples/echoString*.

To import the SDF into LightWave Server using LWSCOM, the SDF must be located in the Enscribe filesystem. Copy the SDF from the SOAPam Server VFS to the Enscribe file system using VFSMGR or any other means appropriate for your environment.

```
run vfsmgr -get file /services/samples/echoString/echoString.sdf echosdf !
```

Once the SDF file is available in the Enscribe filesystem, it can be imported into LightWave Server. The syntax of the LWSCOM IMPORT SDF command is:

```
LWSCOM 1-> help import sdf

IMPORT SDF Command

Imports an SDF from a file into the filesystem as a LightWave DICTIONARY and
an API.

  IMPORT SDF <file-name>
         , API-NAME { <api-name> | * } [ ! ]
         , DICTIONARY-NAME { <dictionary-name> | * } [ ! ]

<file-name>

  Filename of the SDF to import.

API-NAME { <api-name> | * } [ ! ]

  The name to be used for the imported API, or '*' to use an API name derived
  from the SDF.

DICTIONARY-NAME { <dictionary-name> | * } [ ! ]

  The name to be used for the imported DICTIONARY, or '*' to use a DICTIONARY
  name derived from the SDF.
```

For example, to import the Echo String Sample SDF, use this command:

```
run lwscom import sdf ECHOSDF, api-name * !, dictionary-name * !
```

Deploy the Imported API as a LightWave Server Service

To complete the migration, deploy the imported API as a LightWave Server service. This can be done from the LightWave Server Console, or using the LWSCOM Utility. To deploy using the Console, follow these steps:

- Sign in to the Console.
- From the side menu, select Services.
- In the upper right corner, select the '+' icon.
- To deploy the Echo String Service, enter the following:
    - Name: “echoStringSample”
    - Description: “This service implements the echoString sample”
    - Version: use default “1.0”

- API: “echoStringSample”
- Access Control Policy: allow-anonymous-access or any other appropriate ACP
- TCP/IP Port Range: leave blank
- Service Base Path: /services/samples/echoString/echoString
- Enabled: checked

Note: it is important that the correct Service Base Path be configured in order for the correct SOAP service endpoint to be deployed. The Service Base Path for SOAPam compatible services is simply the full path of the SOAPam SDF file, without the “.sdf” extension. For example:

SOAPam SDF location	Service Base Path
/services/samples/echoString/echoString.sdf	/services/samples/echoString/echoString
/services/transactionControl/ transactionControl.sdf	/services/transactionControl/ transactionControl

Stats Configuration

Support for SOAPam Server Statistics Collection has been added to LightWave Server. To enable this feature, use the -*-statscfg* startup option. The feature behavior and configuration file syntax are identical to SOAPam Server.

Command Line Reference

-statscfg <filename>

Specifies the stats configuration file name. If this option is omitted, then no stats configuration file is used, and statistics are not collected.

Configuration Reference

[stats]

enabl ed	The value may be "0" to disable statistics collection or "1" to enable it. The default is "0".

file	The name of the statistics file. If not fully qualified then the value of the _DEFAULTS define is used to complete the file name.
time	Specifies the time format for the statistics file entries. Available formats are: GMT – Greenwich mean time (the default). LST – Local standard time (local time without DST adjustment). LCT – Local civil time (local time with DST adjustment).
flush-interval	Specifies how often statistics are written to the statistics file, specified in seconds, or as a CRON expression. The default value is 15 seconds. Valid values are 15 - 86400.
reset-interval	Specifies how often the process statistics counters are reset, specified in minutes, or as a CRON expression. The default value is 0, which indicates that the statistics are not automatically reset and continue to accumulate for the life of the process. Valid values are 0 - 10080.
on-reset	The action to take on the statistics file when a reset occurs. The available options are 'rollover', 'purgedata', and 'keep'. rollover - The current file is renamed according to the 'rollover-file' specification and a new file is opened. purgedata - The current file data is purged with a call to CONTROL 20 and the current file remains open. keep - No action is taken. Statistics continue to be written to the current file. The default action is 'keep'.

Example

```
; Enable statistics collection.
[stats]
enabled=1
file=$data1.stats.zzstats
time=lct
flush-interval=60
```

Defining Intervals Using CRON Expressions

CRON is a job scheduling service available on a number of platforms. Interval values may be specified as CRON expressions which provide finer grain control on interval expiration. Documenation and tools for creating and interpreting CRON expressions may be found here:

CRON Expressions at Wikipedia

CRON Expression Generator and Explainer at FREEFORMATTER.COM

Example

```
; Enable statistics collection.
[stats]
enabled=1
file=$data1.stats.zzstats
maxentents=200
time=lct
;Flush every 2 minutes
flush-interval=0 */2 * ? * *
; Reset every 4 hours
reset-interval=0 0 */4 ? * *
```

Appendix B: The Authorization Exits Feature

These pages describe the Lightwave Server Authentication Exits feature.

Note: The Authorization Exits feature is licensed separately from the base LightWave Server license. To acquire a license for this feature, please contact your NuWave account representative.

LWS Authentication Exits

Creating a Custom Token Server

Configuring LWS Auth Exits

LWS Authentication Exits

The LightWave Server Authentication Exits feature allows customers to verify custom authentication tokens retrieved from the HTTP headers or queryParams in a REST request.

AuthExits are performed if an AuthExit Rule is defined in an Access Control Policy assigned to the Service associated with the REST request.

- The AuthExit rule's properties are configured much like other Access Control Policy rules, on the Console Access Control page. An AuthExit rule can be added to a new or an existing ACP.
- A Service is configured to use a new or existing ACP that has an AuthExit rule.
- The SERVER/SWORKER process uses the Token Set Configuration defined by the AuthExit Rule to retrieve information about a single named or unnamed token set. A token set is one or more tokens to be used to verify each RESET request. A named token set has a customer provided name.
- The SERVER/SWORKER process manages retrieving token set information, verifying REST request content using the token set information, caching the token set information and verified tokens, and fetching new token set information when the cached token set information expires. When cached verified tokens expire, the tokens from the next REST request are used in a new verification request and, if verified, cached.
- Any token that can be supplied as an HTTP header or query param, such as OAuth bearer tokens and JSON Web Tokens (JWT), may be in a token set.

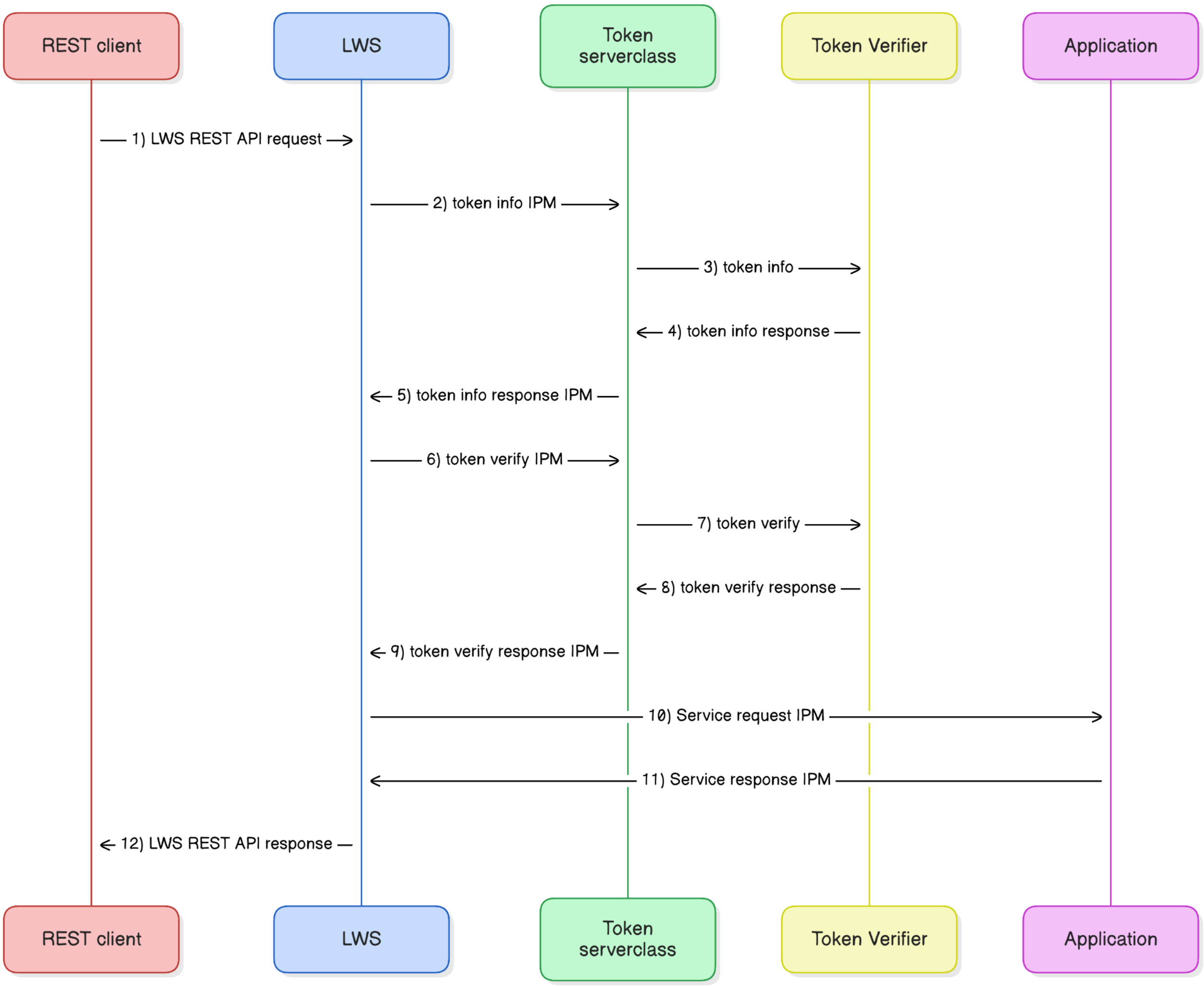
How it Works

When the SERVER/SWORKER process receives a REST request, if cached token set information is available. If not, SERVER/SWORKER queues the request and sends a token set information request to the customer's token server to get new token set information.

- If an error occurs retrieving the token set information, the SERVER/SWORKER process will retry the request up to the limit defined in the AuthExit rule.
- If the token set information is retrieved successfully, the information will be used to retrieve token values from headers and/or queryParams in the REST request. Token vales retrieved from the request are optionally reformatted and base64 decoded before being added to the token verification request.
- For example, a header or queryParam value could be the string "Bearer dGhpcyBpcyB0aGUgdG9rZW4=". The token set info could specify a format string of "Bearer %s" to extract the actual token from the string, and then optionally base64 decode the string "dGhpcyBpcyB0aGUgdG9rZW4=" to 32 bytes of binary data. The resulting data is copied to the token verification request.
- After retrieving token values, as defined by the token set information, from the REST request, and cached verified token values are not available, the SERVER/SWORKER process sends a token verification request to the customer's token server. If cached verified token values are available, the SERVER/SWORKER process compares the verified values in cache with the request's token values.

- If tokens are verified, the REST request is processed as normally. If the token values are not verified, then the REST request is denied.

LWS Request/Response Sequence



Components

- **REST client**: requests services.
- **LWS**: An instance of LightWave Server with the Auth Exits feature enabled.
- **Token serverclass**: A Pathway serverclass that implements the auth exit server.
- **Token Verifier**: The entity that verifies the tokens extracted from the request by LWS.
- **Application**: The backend application providing the service.

Note: The Token serverclass & Token Verifier could be the same entity or a hybrid of serverclass and 3rd party authentication services.

Process Flow

1. A *LWS REST API request* is sent to **LWS** by a **REST client**. The request may contain one or more token values that must be verified before being sent to the **Application**.
2. **LWS** checks its token cache to determine if the token(s) are present for the request. If so, **LWS** moves on to step 6. Otherwise, it sends a *Token Info IPM* to the **Token serverclass**.
3. A *Token Info request* is sent to the **Token serverclass** when **LWS** must retrieve information about one or more tokens that could be present in a request.
4. The *Token Info response* returned by the **Token Verifier** contains the information required for **LWS** to extract the tokens from the request.
5. **LWS** uses the information in the *Token Info response IPM* to extract the tokens from the message in preparation for token verification.
6. **LWS** sends a *Token Verify IPM* to the **Token serverclass** containing the extracted tokens.
7. The *Token Verify request* is sent to the **Token Verifier**.
8. The **Token Verifier** returns the results of the verification in a *Token Verify response*.
9. **LWS** receives the *Token Verify response IPM* from the **Token serverclass**. If the response contains an error, a denial is returned to the **REST Client**.
10. If tokens are verified, **LWS** sends the *Service request IPM* to the **Application**.
11. The **Application** returns the *Service response IPM* to **LWS**.
12. The *LWS REST API response* is returned to the **REST Client**.

Error Handling

When the SERVER/SWORKER process encounters an error, either when communicating with the token server, or an error is returned by the token server itself, it will retry the request using the retry interval and max count supplied in the AuthExit rule of the Service's ACP. If the error condition persists, the error will be logged, and the REST request will be denied with HTTP status 403.

Creating a Custom Token Server

The SERVER/SWORKER process communicates with the token provider serverclass using a standard request and response IPM. The DDL for the message exchange is provided in the *LWAEDDL* file included with the LightWave Server distribution. This DDL can be compiled and used to produce IPM structures in the programming language used for the token server.

The Token Set Information Request

The SERVER/SWORKER process sends the following IPM request to the token server to retrieve token information, where:

- The RQ-CODE field will contain the value LW-AE-RQ-INFO-TKN-SET-V1.
- The RQ-TS-UNIQUE field will contain a unique value retrieved from the TS_UNIQUE_CREATE_ system procedure call.
- The TKN-SET-NAME and TKN-SET-NAME-LEN are filled in from the values in the AuthExit rule, if the tokenSetName property is defined in the rule. If the property is not defined, the string field will be filled with the configured string padding character, and the binary value set to zero.

All other fields will be filled with binary 0s.

```
Definition LW-AE-INFO-TKN-SET-V1-RQ.
  Num  LV Element Name                         Offset(Bit) Data Type & Size
Source Definition
  001  01 LW-AE-INFO-TKN-SET-V1-RQ             0           Group 292
  002   02 HEADER                              0           Group
32                                   LW-AE-RQ-HEADER
  003    03 RQ-CODE                            0           Enumeration
    004     89 LW-AE-RQ-FETCH-TKN-SET-V1                   Enumeration clause
    005     89 LW-AE-RQ-INFO-TKN-SET-V1                    Enumeration clause
    006     89 LW-AE-RQ-VERIFY-TKN-SET-V1                  Enumeration clause
  007    03 RESERVED-1                         2           Binary(16) signed
  008    03 RQ-TS-UNIQUE                       4           Binary(64) signed, Occurs
[2:2]
  009    03 RESERVED                           20          Character 12
  010   02 TKN-SET-NAME-LEN                    32          Binary(32) signed
  011   02 TKN-SET-NAME                        36          Character
256                              LW-AE-TKN-SET-NAME
          Definition size is 292 bytes.
```

The Token Set Verification Response

Upon receiving this request, the token server should verify that the values sent in the request are valid. The outcome is indicated in the IPM reply where:

- The RP-CODE indicates the outcome of the request:
    - LW-AE-RP-SUCCESS indicates that all the token values from the verification request are valid.
    - LW-AE-RP-RETRY indicates that the SERVER/SWORKER process should retry the request.

  - LW-AE-RP-ERROR indicates that an error occurred with the Token Server, or that one or more of the token values are invalid. The result means the same for the token verification of the REST request.
- If RP-CODE is LW-AE-RP-SUCCESS:
  - The set of token values in the request are valid.
  - TKN-SET-TTL is the number of seconds for which the values may be retained in the token cache. Future REST requests are verified against the token cache until the TTL has expired. Once expired, the next REST request will result in a new verification request sent to the Token Server.
- If RP-CODE is LW-AE-RP-ERROR:
  - LWS attempts to retry the token verification request until RP-CODE is LWS-AE-RP-SUCCESS or LWS exhausts its configured retry attempts.
  - If retry attempts are exhausted, the current (and any pending) REST request awaiting a token verification are denied with HTTP status 403.
- TKN-SET-NAME and TKN-SET-NAME-LEN must echo the values from the verification request.

```
Definition LW-AE-INFO-TKN-SET-V1-RP.
  Num  LV Element Name                        Offset(Bit) Data Type & Size
Source Definition
  001  01 LW-AE-INFO-TKN-SET-V1-RP              0           Group 8716
  002   02 HEADER                               0           Group
32                                  LW-AE-RP-HEADER
  003    03 RP-CODE                             0           Enumeration
    004     89 LW-AE-RP-SUCCESS                             Enumeration clause
    005     89 LW-AE-RP-RETRY                               Enumeration clause
    006     89 LW-AE-RP-ERROR                               Enumeration clause
  007    03 RETRY-INTERVAL                      2           Binary(16) signed
  008    03 RESERVED                            4           Character 28
  009   02 TKN-SET-NAME-LEN                     32          Binary(32) signed
  010   02 TKN-SET-NAME                         36          Character
256                             LW-AE-TKN-SET-NAME
  011   02 TKN-SET-TTL                          292         Binary(32) unsigned
  012   02 TKN-COUNT                            296         Binary(32) signed
  013   02 TKN-SET                              300         Group 526, Occurs [1:16]
LW-AE-INFO-ITEM-V1
              Occurs depends on TKN-COUNT.
  014    03 TKN-TYPE                            300         Enumeration
    015     89 LW-AE-TKN-TYPE-HEADER                        Enumeration clause
    016     89 LW-AE-TKN-TYPE-PARAM                         Enumeration clause
  017    03 BASE64-DECODE                       302         Binary(32) signed
  018    03 TKN-FORMAT-LEN                      306         Binary(32) signed
  019    03 TKN-FORMAT                          310         Character 256
  020    03 TKN-NAME-LEN                        566         Binary(32) signed
  021    03 TKN-NAME                            570         Character 256
```

```
          Definition size is 8716 bytes.
```

The Token Set Verification Request

The SERVER/SWORKER process sends the following IPM request to the token server to verify token values, where:

- The RQ-CODE field will contain the value LW-AE-RQ-VERIFY-TKN-SET-V1.
- The RQ-TS-UNIQUE field will contain a unique value retrieved from the TS_UNIQUE_CREATE_ system procedure call.
- The TKN-SET-NAME and TKN-SET-NAME-LEN are filled in from the values in the AuthExit rule, if the tokenSetName property is defined in the rule. If the property is not defined, the string field will be filled with the configured string padding character, and the binary value set to zero.
- TKN-COUNT is the number of token values in the request.
- From 1 to 16 token values may be present in TKN-SET:
  - TKN-TYPE is one of LW-AE-TKN-TYPE-HEADER or LW-AE-TKN-TYPE-PARAM.
  - The TKN-NAME and TKN-NAME-LEN fields contain the token name and the token name length, respectively. The values are copied from the token info set returned by the Token Server.
  - The TKN-VALUE and TKN-VALUE-LEN fields contain the value and value length, respectively, of the token extracted from the named header or query param in the REST request.

Note that the verification request message length will vary depending on the number of tokens sent.

```
Definition LW-AE-VERIFY-TKN-SET-V1-RQ.
  Num  LV Element Name                        Offset(Bit) Data Type & Size
Source Definition
  001  01 LW-AE-VERIFY-TKN-SET-V1-RQ            0           Group 37128
  002   02 HEADER                               0           Group
32                                  LW-AE-RQ-HEADER
  003    03 RQ-CODE                             0           Enumeration
    004      89 LW-AE-RQ-FETCH-TKN-SET-V1                   Enumeration clause
    005      89 LW-AE-RQ-INFO-TKN-SET-V1                    Enumeration clause
    006      89 LW-AE-RQ-VERIFY-TKN-SET-V1                  Enumeration clause
  007    03 RESERVED-1                          2           Binary(16) signed
  008    03 RQ-TS-UNIQUE                        4           Binary(64) signed, Occurs
[2:2]
  009    03 RESERVED                            20          Character 12
  010   02 TKN-SET-NAME-LEN                     32          Binary(32) signed
  011   02 TKN-SET-NAME                         36          Character
64                              LW-AE-TKN-SET-NAME
  012   02 TKN-COUNT                            100         Binary(32) signed
  013   02 TKN-SET                            104         Group 2314, Occurs [1:16]
LW-AE-VERIFY-ITEM-V1
              Occurs depends on TKN-COUNT.
  014    03 TKN-TYPE                          104         Enumeration
    015     89 LW-AE-TKN-TYPE-HEADER                      Enumeration clause
    016     89 LW-AE-TKN-TYPE-PARAM                       Enumeration clause
  017    03 TKN-NAME-LEN                      106         Binary(32) signed
  018    03 TKN-NAME                          110         Character 256
  019    03 TKN-VALUE-LEN                     366         Binary(32) signed
  020    03 TKN-VALUE                         370         Character 2048
          Definition size is 37128 bytes.
```

The Token Set Verification Response

Upon receiving this request, the token server should verify that the values sent in the request are valid. The outcome is indicated in the IPM reply where:

- The RP-CODE indicates the outcome of the request:
  - LW-AE-RP-SUCCESS indicates that all the token values from the verification request are valid.
  - LW-AE-RP-RETRY indicates that the SERVER/SWORKER process should retry the request.
  - LW-AE-RP-ERROR indicates that an error occurred with the Token Server, or that one or more of the token values are invalid. The result means the same for the token verification of the REST request.
- If RP-CODE is LW-AE-RP-SUCCESS:
  - The set of token values in the request are valid.
  - TKN-SET-TTL is the number of seconds for which the values may be retained in the token cache. Future REST requests are verified against the token cache until the TTL has expired. Once expired, the next REST request will result in a new verification request sent to the Token Server.
- If RP-CODE is LW-AE-RP-ERROR:
  - LWS attempts to retry the token verification request until RP-CODE is LWS-AE-RP-SUCCESS or LWS exhausts its configured retry attempts.
  - If retry attempts are exhausted, the current (and any pending) REST request awaiting a token verification are denied with HTTP status 403.
- TKN-SET-NAME and TKN-SET-NAME-LEN must echo the values from the verification request.

```
Definition LW-AE-VERIFY-TKN-SET-V1-RP.
  Num  LV Element Name                        Offset(Bit) Data Type & Size
Source Definition
  001  01 LW-AE-VERIFY-TKN-SET-V1-RP          0           Group 104
  002   02 HEADER                             0           Group
32                                  LW-AE-RP-HEADER
  003    03 RP-CODE                           0           Enumeration
    004     89 LW-AE-RP-SUCCESS                           Enumeration clause
    005     89 LW-AE-RP-RETRY                             Enumeration clause
    006     89 LW-AE-RP-ERROR                             Enumeration clause
  007    03 RETRY-INTERVAL                    2           Binary(16) signed
  008    03 RESERVED                          4           Character 28
  009   02 TKN-SET-NAME-LEN                   32          Binary(32) signed
  010   02 TKN-SET-NAME                       36          Character
64                              LW-AE-TKN-SET-NAME
  011   02 TKN-SET-TTL                        100         Binary(32) unsigned
          Definition size is 104 bytes.
```

Configuring LWS Auth Exits

LightWave Server Authentication Exits are configured with an Access Control Policy AuthExit rule, similar to the configuration of Identity, Source IP Address, and CORS Origins rules. The difference between AuthExit rules and the other types of rules is that there can be only one AuthExit rule per Access Control Policy.

The AuthExit rule is activated by its ACP being assigned to a LWS Service, just like any other rule type.

Property	**Description**	**Default**
Pathmon	The process name of the serverclass PATHMON process controlling the token server. May be specified as a DEFINE.	none required
Serverclass	The serverclass name of the token server.	none required
Token Set Name	The name of the token set for this AuthExit rule. If no token set name is provided, it is up to the token server as to which token set will be used.	none
String Padding	Determines whether string fields sent to the token server are padded with spaces (*spaces*) or binary zeros (*zeros*).	zeros
Retry Interval	The number of seconds to wait after an error has occurred before attempting another request to the token server.	3

Property	Description	Default
Retry Maximum	The number of retries to attempt before returning an error to the REST client.	3

Appendix C: The Password Policy Feature

This Appendix describes the password policy feature in LightWave Server. The Password Policy feature is available for all LightWave Server customers at no additional license cost.

Introduction to the Password Policy feature

Configuring Password Policy

Introduction to the Password Policy feature

The Password Policy feature enables LightWave Server customers to enforce rigorous password quality standards.

How it Works

LightWave Server's Password Policy utilizes a background refresh mechanism to detect policy revisions automatically, maintaining system uptime by avoiding manual service restarts.

- Where appropriate, the UI or the console checks against the policy to ensure password quality.
    - Some examples of password changes:
        - Adding a user to LightWave Client or Server
        - Changing the administrator password.
- The Password Policy is enforced for all new users or when setting a new password for an existing user.
- The Password Policy defined by the customer is also enforced for non-console user IDs when used for request authorization.
- The Password Policy configuration is set by a startup option:

Configuring Password Policy

The Password Policy is defined in a YAML text file. The policy to use is set with the `--password-policy` startup option. For example:

```
run server / <run-options> / --password-policy pwdpol <other-options>
```

In this case, PWDPOL is the name of the Password Policy YAML configuration file.

This is an example policy file:

```
PasswordPolicyConfig:
  monitorInterval: 15
  loginOptions:
    maxFailures: 5
  passwordOptions:
    minLength: 8
    maxLength: 12           # Absolute maximum is 64
    special: true           # at least one special character
    digit: true             # at least one numeric digit
    upper: true             # at least one uppercase letter
    lower: true             # at least one lowercase letter
    firstMustBeLetter: false # first character must be alpha
  version: 1
```

Element	Value	Notes
`monitorInterval`	How often, in seconds, to check for a change in the YAML configuration file.	Values: `5 - 65535` Default: `5`
`maxFailures`	The allowed maximum number of login failures before the account is disabled (locked). Disabled accounts must be manually enabled.	Values: `1 - 60` Default: `5`
`minLength`	The minimum length for a password.	Values: `1 - 64` Default: `1`
`maxLength`	The maximum length for a password.	Values: `1 - 64` Default: `64` Must be greater than `minLength`.

Element	Value	Notes
`special`	Password must have at least one special character?	Values: `true` , `false` . Default: `false`
`digit`	Password must have at least one numeric digit.	Values: `true` , `false` . Default: `false`
`upper`	Password must have at least one uppercase alphabetic character.	Values: `true` , `false` . Default: `false`
`lower`	Password must have at least one lowercase alphabetic character.	Values: `true` , `false` . Default: `false`
`firstMustBeLetter`	The first character of the password must be an uppercase or lowercase alphabetic character.	Values: `true` , `false` . Default: `false`

How to Obtain Support

We welcome your feedback and encourage you to contact us with any questions or comments. Please visit the NuWave Technologies Support Center to learn how to contact us.

We work hard to deliver quality products to our customers, but occasionally something goes wrong. Please review our suggestions for How to Submit a Product Support Request. Below are instructions for obtaining product-specific details you should include in your Support Request:

Determine the Product Version

Use the NonStop VPROC utility to find the LightWave Server product version:

```
> VPROC [<installation-subvolume>.]SERVER
```

Copy and paste the VPROC output to your support request.

Include a Copy of the Dictionary and API Definition

If your issue concerns a specific API that you are using, attach a copy of the dictionary and API to your Support Request. You can obtain a copy of your dictionary by using the Console. Navigate to the dictionary at issue. In the dictionary view toolbar, click the (down-arrow icon) to export the dictionary to a file. You can obtain a copy of your API by using the Console. Navigate to the API at issue. In the dictionary view toolbar, click the (down-arrow icon) to export the API to a file.

It is often helpful to include a copy of the DDL source that was used to create the DDL dictionary that your server application was built from.

Enable Diagnostic Logging

A diagnostic log includes detailed information about a single API method invocation. This information can be extremely useful in diagnosing problems. If your issue concerns a specific API, enable diagnostic logging. Reproduce your issue so that diagnostics can be logged. Download the diagnostic log and attach it to your Support Request.

View as PDF

To view and/or download a PDF version of this document, click the link below.

LightWave Server Release 1.3.0 Documentation

LightWave Server Documentation

Release 1.3.0

Document
Classification: Proprietary and Confidential

Type: Customer Documentation

Author: NuWave Technologies, Inc.

Document Version: 1.3.0

Release Date: 26 Jan 2026

Contact: NuWave Technologies

Address 301 Edgewater Pl, STE 100, Wakefield, MA 01880

Email: support@nuwavetech.com

Web: https://support.nuwavetech.com/support/home

support@nuwavetech.com