NuWave
MISSION-CRITICAL ENTERPRISE INTEGRATION

Product Documentation

LightWave Client Documentation

LightWave Client Release 1.3.1
15 July 2025

Table of Contents

Welcome to the documentation for LightWave Client™ 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

How To Begin

Introduction to LightWave Client

For Developers

Getting Started

Working with APIs

Deploying APIs

For System Administrators

Administrator Guide

Installing LightWave Client

Release Notes

Appendices

Appendix A: The SOAPam Server Compatibility Feature

How To Begin

This section describes how to proceed if you are a first time LightWave Client user. LightWave Client 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 Client processes. Systems/Operations managers usually fall into this category.
- Developers - A developer is anyone responsible for building applications that use LightWave Client. Developers often need to manage LightWave Client 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 Client.
- Install LightWave Client as described in Installing LightWave Client.
- Start LightWave Client as described in Starting and Stopping LightWave Client.

Developers

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

Others

- Read section Introduction to LightWave Client.

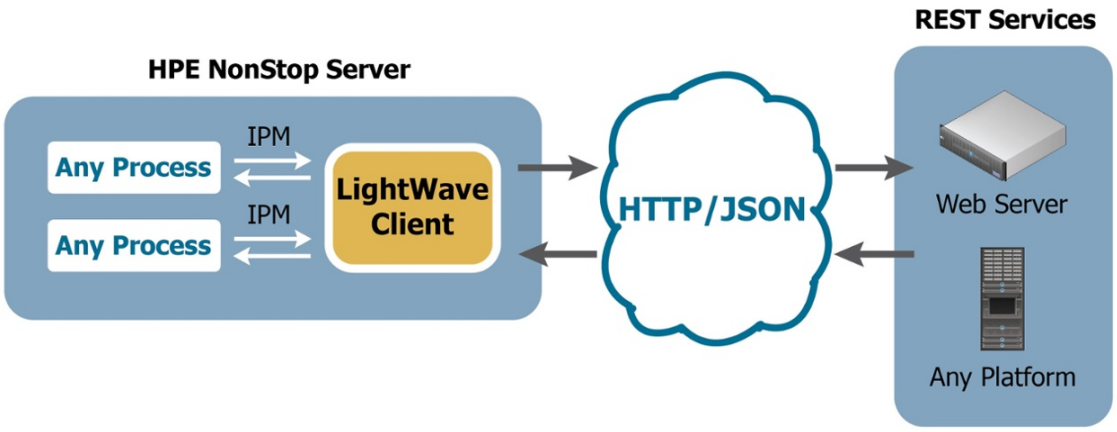
Introduction to LightWave Client

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

What is LightWave Client

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 Client allows applications running on NonStop servers to securely access REST Web services anywhere, on any platform or operating system. Your NonStop application simply sends a formatted interprocess message to the LightWave Client Process which handles all of the details of mapping the IPM elements to a JSON or XML message, exchanging the message with the Web service provider, and parsing the JSON or XML response into the reply IPM. The LightWave CLIENT process hides the complexity of the TCP/IP, HTTP, SSL/TLS, JSON, and XML protocols required to access REST Web services.



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

- Use services provided by business partners to check inventory, place orders, obtain delivery schedules, etc.
- Integrate NonStop applications with ERP or CRM systems
- Access real-time securities pricing, currency exchange rates, and other time-sensitive data
- Incorporate centralized authentication and authorization services

Product Components

LightWave Client consists of two major component processes: Console and Client.

Console

The role of the Console process is to support the administrative and management user interface for LightWave Client. Console provides a browser interface which allows you to:

- View Console status
- Manage APIs
- View Client diagnostic logs
- View Console HTTP logs
- Manage SSL/TLS Server Certificates (for Console)
- Manage LightWave Client users and groups (for access to Console)

Once the Console process has been started, you access the LightWave Console application using your web browser.

Client

The role of Client is to provide the runtime implementation of a LightWave Client service you defined earlier using Console. Although you can run one or more client processes for a given service (e.g., for load balancing), each instance of the Client process supports exactly one service. The Client process accepts an interprocess message on $RECEIVE from a client application, which it transforms into a REST web-API request and forwards to a service endpoint using HTTP over TCP/IP. When the service responds, Client transforms the response into an interprocess message which it REPLYs to the original client requester application. Each Client process is capable of supporting simultaneous requests from multiple client applications.

Security

LightWave Client provides security features to ensure that your applications and NonStop Server remain secure.

Transport Security

LightWave Client uses OpenSSL to provide transport security using the latest version of TLS. Client applications can take advantage of this feature by configuring the Client process' `baseUrl` parameter to use the HTTPS protocol.

User Authentication

LightWave Client supports HTTP Basic/Digest authentication and SSL mutual authentication with web service endpoints.

LightWave Client provides security features to ensure that your applications and NonStop Server remain secure.

Transport Security

LightWave Client uses OpenSSL to provide transport security using the latest version of TLS. Client applications can take advantage of this feature by configuring the Client process' `baseUrl` parameter to use the HTTPS protocol.

User Authentication

LightWave Client supports HTTP Basic/Digest authentication and SSL mutual authentication with web service endpoints.

Performance and Scalability

LightWave Client was designed to be fully scalable using NonStop Pathway. The CLIENT process is multi-threaded and can process multiple simultaneous requests. CLIENT can also be configured as a Pathway serverclass, allowing it to scale to match demand like any other Pathway server.

Developer Guide

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

> **Tip**
>
> Visit the NuWave GitHub site for sample applications.

Getting Started

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

LightWave Client provides access to REST APIs through the familiar NonStop interprocess message system. Your client application constructs an interprocess message (IPM) formatted according to a definition generated by LightWave Client. Your application then sends the IPM (using COBOL "SEND", or the SERVERCLASS_SEND_ or WRITEREAD[X|64] system procedures) to the LightWave Client Process, which transforms the IPM content into an HTTP message formatted as required by the REST/web API and sends it to the service endpoint. When the service completes, LightWave Client receives the response and transforms the content into an IPM which is then replied to your client application.

Client applications can be coded in any programming language supported on HPE NonStop.

The maximum interprocess message size supported by LightWave Client 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.

These are the steps required to develop a client application:

- Using the Console, create the API definition for the service you wish to access:
  - Import a definition in OpenAPI (a/k/a Swagger) format that is provided by the service owner, or
  - Create the definition using the API Editor together with sample requests and responses provided by the service owner
- Generate the API interface IPMs to be used by your application
  - Using the Console, generate DDL source for the IPMs
  - Load generated DDL source into a DDL dictionary
  - Generate language-specific header files
- Create code in your application to call the service using the IPMs defined in the header files.
  - Include the generated header file
  - set the 'method alias' and any API-specific parameters
  - send the request IPM to the LightWave Client Process
- Run the application
  - Configure and start the LightWave Client Process Instance
  - Run your client application

Working with APIs

Overview

An API defines the REST or web interface to services provided by a service owner. In order to use a REST service, the owner must provide you with some type of documentation for the service interface, either through an OpenAPI (also known as 'Swagger') API definition, or written descriptions of the paths, methods, parameters and body content required to use the service.

> In this document, we use the term "REST service" to refer to web services offering REST or web APIs over HTTP, whether or not they fully conform to REST (REpresentational State Transfer; Roy Fielding) conventions. You may find owners of these services to use different terminology, sometimes inaccurately, to refer to the same thing: "REST service", "REST API", "web API", etc.

LightWave Client requires an API definition in order to facilitate access from client applications on NonStop servers to remote REST services. You use the LightWave Client Console browser application to create, view and edit API definitions. You can create an API definition by importing a Swagger definition or using the Console's API Editor together with example requests and responses, either of which may be provided by the service owner. You may also import an API definition previously created and exported from LightWave Client.

API Definitions

An API *definition* consists of

- an API name
- a description
- a list of operations

An *operation*, in turn, is comprised of

- a URI path (e.g. /employees/{employee-id}
- one or more methods

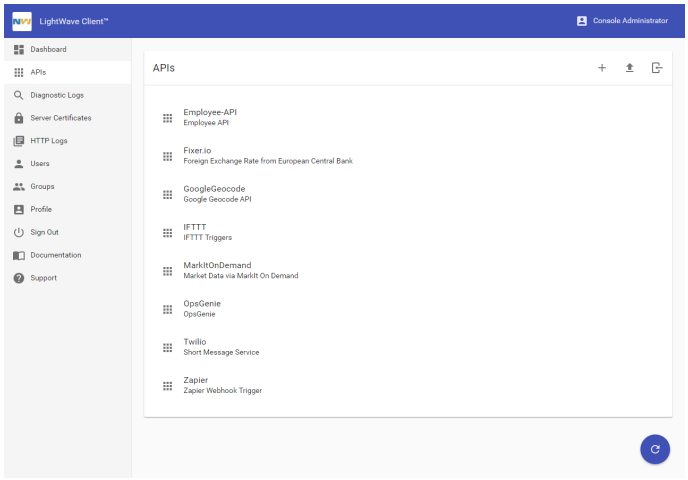
A *method* is comprised of

- an HTTP verb (GET, POST, etc.)
- a textual description (optional)
- I/O options
- a request definition
- one or more response definitions

Each request and response definition may include zero or more mappings. The request mappings determine how LightWave Client transforms data received in the client application request IPM into the HTTP request that will be sent to the remote service endpoint. Likewise, response mappings determine how LightWave Client transforms the data in the HTTP response from the service endpoint into the IPM that will be replied to the client application. Since the REST API may return differently formatted responses depending on the outcome of the request, each method may include more than one response definition.

Using the Console

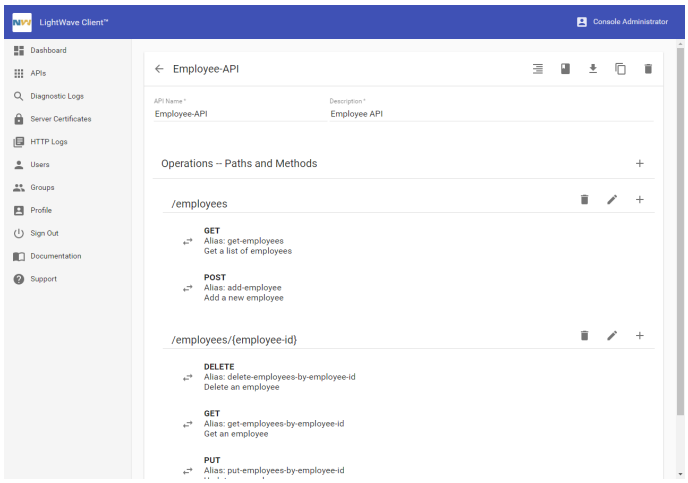
To begin using the LightWave Client Console to work with APIs, sign in to the Console, then use the menu to navigate to APIs. The list of existing APIs is displayed.



Use the view menu in the upper right corner to:

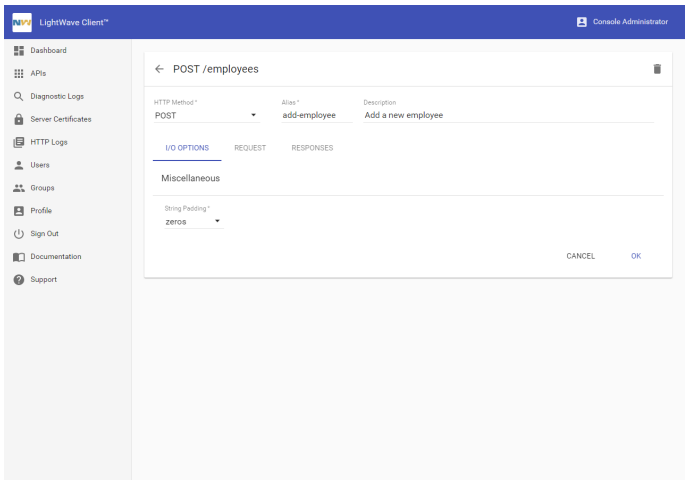
- + Create an API using the Editor.
- Import an API previously created and exported by LightWave Client.
- Create an API by importing a Swagger definition.

To view the definition of an API, click the name of the API to open the API Editor for the selected API.

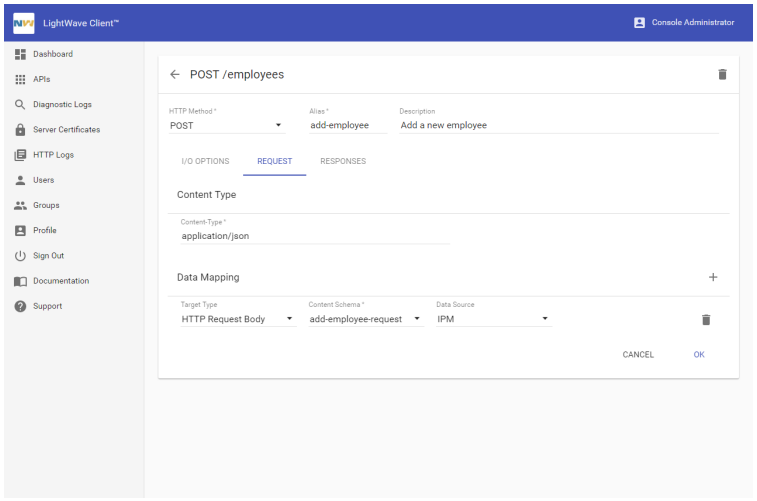


Using the API Editor

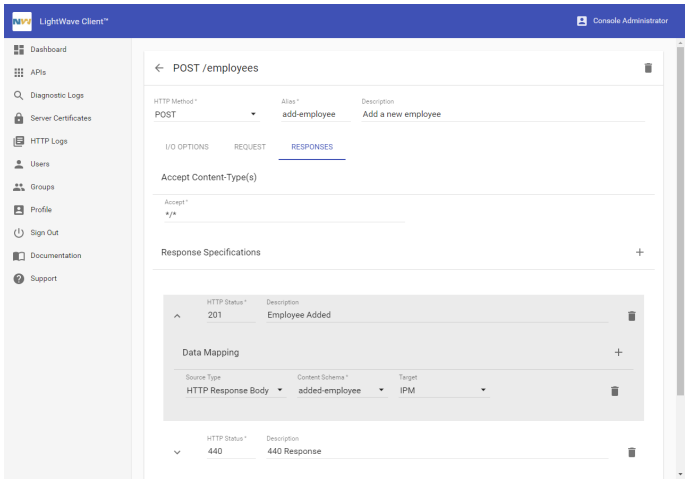
The API Editor is used to create, view and modify API definitions. Each API consists of one or more operations. An operation is the combination of a path and a method (an HTTP verb, e.g. GET or POST). An operation defines an HTTP request and response.



Operation I/O Options



Operation Request Definition



Operation Response Definition

Each operation is defined to have a single request format, for which multiple response formats can be defined. Each request and response can include zero or more data item mappings.

I/O Options

StringPadding specifies how LightWave will treat character string fields in interprocess messages (IPMs).

setting	description
zeroes	A setting of zeros means that LightWave will expect the client application to zero-terminate (or "null-terminate") strings in IPMs it sends to LightWave, and LightWave will zero-terminate strings in IPMs it returns to the client application. This setting is typical for client applications written in the C/C++ programming language.
spaces	A setting of spaces means that LightWave will expect the client application to blank pad strings in IPMs it sends to LightWave, and LightWave will blank pad strings in IPMs it returns to the client application. This setting is typical for client applications written in the COBOL programming language.

Request

Content Type

Content Type specifies the value of the Content-Type HTTP header that will be sent with the request to the web service. The value should be a media type name (see Media Types).

If the request specifies an HTTP Request Body mapping (see below) which is set to a schema type other than "BLOB", the media type should be "application/json" to indicate a JSON payload or "text/xml" to indicate an XML payload. If the schema type is "BLOB", set the value to a media type that describes the content of the BLOB, which should be a type that the web service accepts. If the request specifies one or more HTTP Form Field mappings, the value should be "application/x-www-form-urlencoded".

Data Mapping

LightWave performs mapping of data items from interprocess messages (IPMs) that client applications send to various components in the HTTP request that is sent to the web service. Depending on the component of the HTTP Request being mapped to, there are additional parameters required.

target	description	name	source
URI Path Component	a component of the path for the current method; where the path has been defined to contain one or more 'path parameters', e.g. /employees/{employee-id}, where {employee-id} is a path parameter. At runtime, LightWave will construct the request path by replacing any path parameter placeholders with the corresponding mapping's 'source' value.	the path parameter name	IPM, constant, API parameter
URI Query Parameter	a name=value pair appended to the method's request path, e.g., /employees?format=detailed&sort=ascending, where 'format' and 'sort' are query parameter names.	the query parameter name	IPM, constant, API parameter
HTTP Form Field	a name=value pair that is encoded and combined with other form fields and placed in the request message body "content". This mapping type is mutually exclusive with "HTTP Request Body". See also ContentType, above.	the form field name	IPM, constant, API parameter
HTTP Request Header	a name:value pair that is added to the list of headers in the request message to be sent to the web service	the header name	IPM, constant, API parameter

target	description	name	source
HTTP Request Body	an object in Javascript Object Notation described by a schema. A schema can be added 'manually', or 'by example' using sample request content (typically provided by the web service owner). A maximum of one HTTP Request Body mapping is allowed, and is mutually exclusive with "HTTP Form Field" mappings. Alternatively, the request body content schema can be defined as "BLOB", in which case LightWave inserts the client-application supplied data into the request body verbatim. See also ContentType, above.	a schema (type) name or BLOB	IPM, FILE

source	description
IPM	the source value is taken from a field in the request IPM. The maximum length of the value must also be specified.
constant	the source value is taken from the constant value specified in the mapping
API parameter	the source value is taken from a command line parameter supplied to the LightWave CLIENT process. See Working with API Parameters.
FILE	the source content is taken from a file, the name of which is supplied by the client application in the request IPM

Responses

You may define one or more responses for each request, each response corresponding to an HTTP status code that may be returned from the called web service.

Accept Content Type

Specifies the value of the Accept HTTP header that will be sent with the request to the web service. The value should be a media type name (see Media Types), or a comma-separated list of types. Wildcards are allowed. The default value is ***/***, which is suitable in most cases. However, some web services provide responses in different formats according to

the Accept header (for example, JSON vs. XML). In such cases, it may be necessary to specify a value other than the default ("application/json", for JSON format, for example). See W3C HTTP Header Field Definitions, section 12.5.1 Accept for more information.

Data Mapping

LightWave performs mapping of data items from the called web service's HTTP response into fields in the interprocess messages (IPM) that will be returned to the client application. Zero or more mappings may be defined for each response.

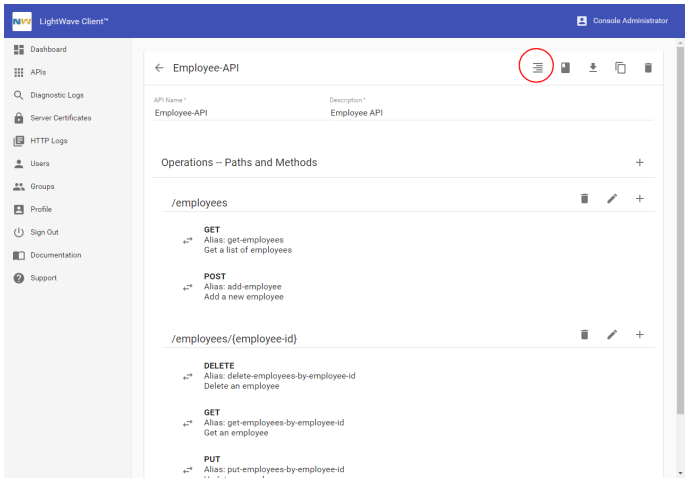
source	description	name	target
HTTP Response Header	a name:value pair in the list of headers contained in the response from the web service. The maximum size of the value must also be specified.	the header name	IPM
HTTP Response Body	an object in Javascript Object Notation described by a schema. A schema can be added 'manually', or 'by example' using sample response content (typically provided by the web service owner). A maximum of one HTTP Request Body mapping is allowed. Alternatively, the request body content schema can be defined as "BLOB", in which case LightWave stores the raw body content in an Enscribe file and copies the name of the file to a field in the IPM that is returned (REPLY-ed) to the client application.	a schema (type) name or BLOB	IPM, FILE

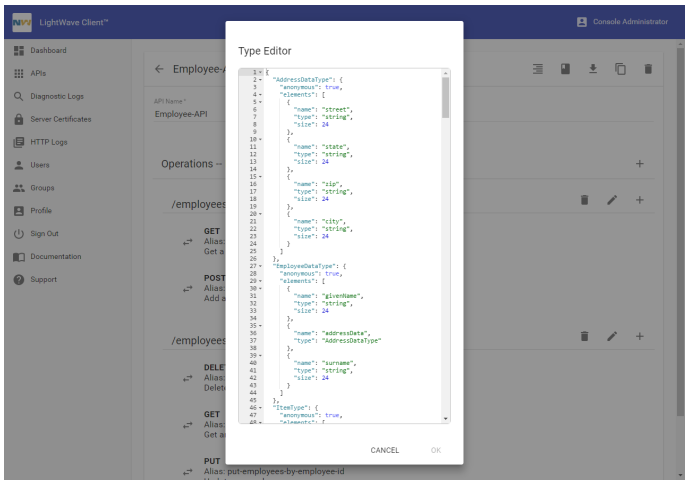
target	description
IPM	the response value is stored in the IPM
FILE	the response body content is stored in a file, the name of which is returned in a field in the IPM. See --blob-files command line parameter for the LightWave CLIENT process for related information.

Importing a Swagger (OpenAPI) Definition

You can create an API by importing a Swagger or OpenAPI definition, either by entering the location of the definition by URL, or by pasting the definition into the text box. LightWave Client translates the Swagger or OpenAPI definition into LightWave Client format. You may choose to edit the imported definition to adjust to your liking the names of certain items (e.g., method aliases or type names) that were generated by LightWave

Annotating Data Types





The request and response message content (or 'body') of REST APIs is composed of Javascript primitive data types (string, number, boolean, null), objects and arrays (which themselves can contain primitives). Certain primitive types do not translate cleanly to NonStop data types. For example, Javascript strings are inherently of unlimited length, whereas NonStop strings must be allocated to have a finite maximum length. To accommodate this, LightWave allows data type descriptions to be 'annotated' with additional properties:

```
"elements": [
```

```
  {
  "name": "state",
  "type": "string",
  "size": 2
  },
  {
  "name": "zipcode",
  "type": "string",
  "size": 9
  }
]
```

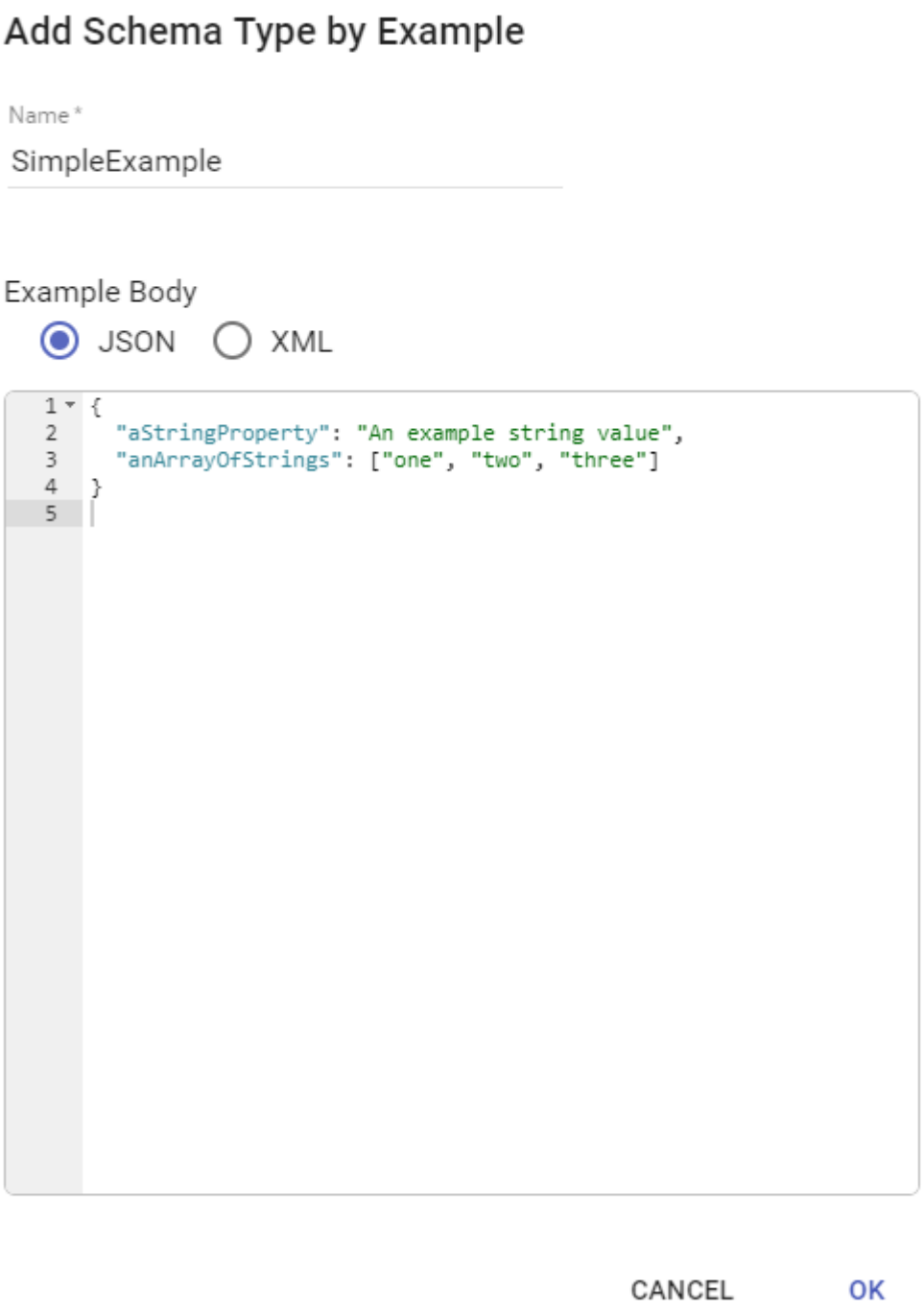
See Working with Schema for information on the available element properties.

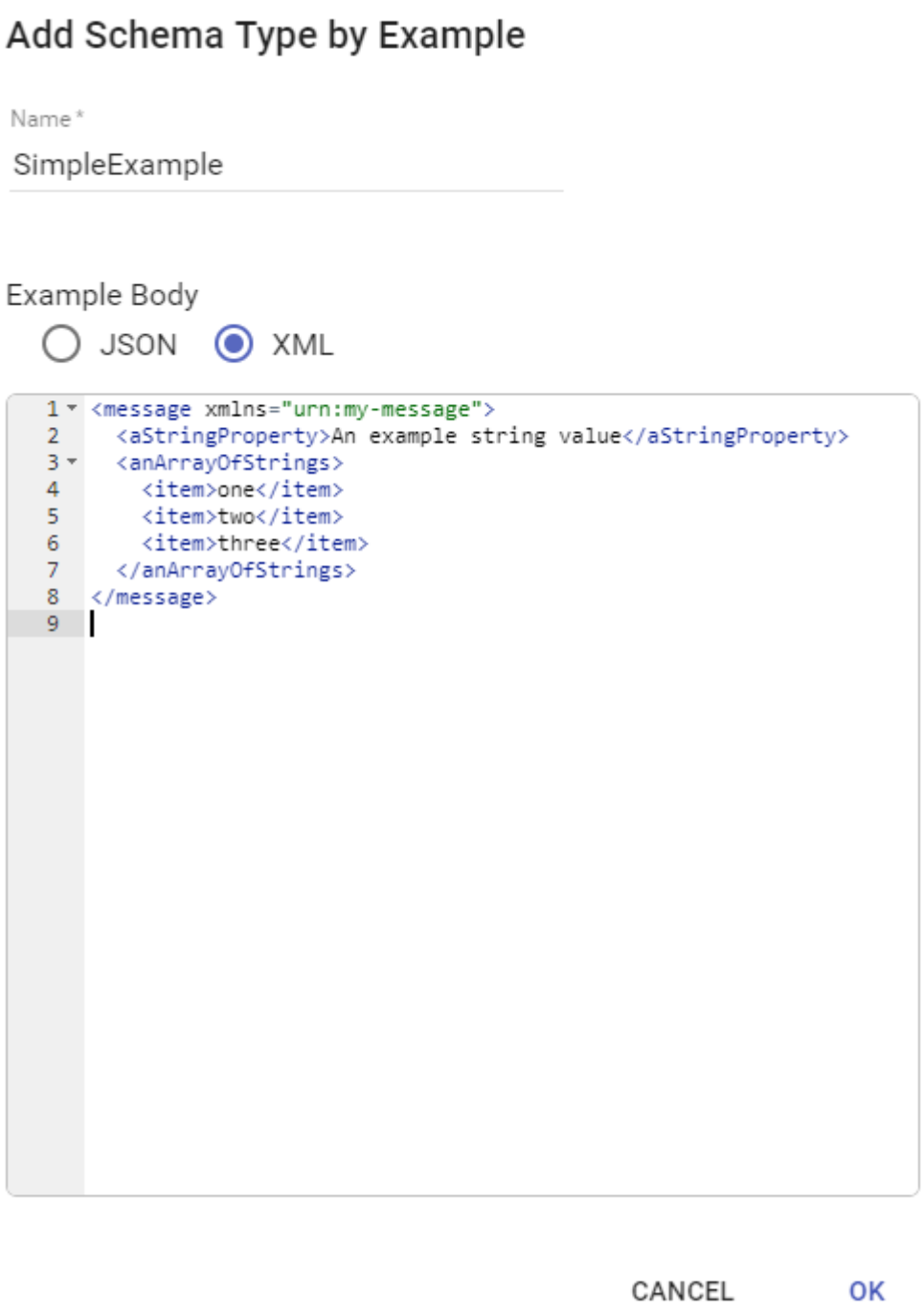
Working with Schema

LightWave Client schema describes the data types used in API request and response data mappings. The schema definitions are stored as part of the API. Ordinarily, there is no need for you to create schema definitions directly. The LightWave Client Console API editor automatically creates schema definitions for you when, while defining an API, you create a request or response mapping and select "Add by Example" for 'Schema Type'. In addition, when importing an API definition from a Swagger or OpenAPI definition, the API editor translates the type definitions it contains to LightWave Client schema.

To view or edit the schema attached to your API, click the 'Edit API Schema' icon in the API detail view toolbar. The schema is defined in Javascript Object Notation (JSON) regardless if data types are serialized as JSON or XML.

Although the Console API editor ordinarily creates the schema for you, there are times that you may want to modify the generated schema. The reason for this is due to differences in JSON schema or XML schema used in REST APIs and the invariant structured data types used in HPE NonStop applications. Both JSON and XML support the concept of unbounded strings – character strings without a defined maximum length. Likewise, both JSON and XML support unbounded arrays – arrays without a defined maximum size. Because the programming languages (C, COBOL, TAL, etc.) used by client applications on NonStop do not support unbounded data structures, it's necessary for LightWave to impose a maximum limit on string length and array size. When the API editor generates schema, it uses a minimum default length for strings of 32 and a minimum default array size of 4. (If the structure provided 'by example' contains a longer string or larger array, the example size is used.) Fortunately, these limits may be modified – on an element-by-element basis – by modifying the schema the Console API editor has generated for you.


Add Schema Type by Example
Name *
SimpleExample
Example Body
JSON
XML
{
"aStringProperty": "An example string value",
"anArrayOfStrings": ["one", "two", "three"]
}
CANCEL
OK



The image above shows the 'Add by Example' dialog. Note the simple example JSON object consisting of two properties, "aStringProperty" and "anArrayOfStrings", and their respective sample values. The image below shows schema generated by the examples above.

Schema Editor

Each schema type definition consists of an array of elements describing the properties or fields of the object it's describing. Each element minimally has a 'name' and a 'type'. See Built-in primitive types, below, for a list of available types. There are additional properties that may be added to an element's schema to further describe the element. See Schema Element Properties, below for a list of those properties.

Note that aStringProperty has been given a default "size", or length, of 32. If we know that aStringProperty will never hold a value greater than 12 characters, we can modify the size property accordingly. Likewise, if we know that a value for aStringProperty may be as long as 100 characters, we must modify the size property accordingly. (If at run time, a web service returns a value longer than ‘size' allows, LightWave will copy 'size' characters to the buffer and return a 'field truncated' warning.) If your client application will use zero-padded or NULL terminated strings as a 'C’ or TAL language program typically would, be sure to

account for the terminator when calculating 'size'. See stringPadding in Schema Element Properties, below.

Regarding anArrayOfStrings in the example, note that again, a default size of 32 has been generated. The size property should be adjusted as needed. Additionally, since anArrayOfStrings is an array, the schema generator has emitted the maxOccurs property and given it the default value of 4, meaning the array may hold a maximum of four elements. If we know that the array must hold more or less than that amount, we would modify the maxOccurs property accordingly. (If the number of array items returned by the web service exceeds maxOccurs, LightWave will return an 'array truncated' warning.) Finally, note the dependsOn property. The schema generator has inserted an 'array count' element named 'anArrayOfStrings-count' into the type definition to hold the logical length of the anArrayOfStrings array. During serialization, only anArrayOfStrings-count items will be serialized to JSON. During deserialization, LightWave will store the actual number of items converted from JSON and copied into anArrayOfStrings in anArrayOfStrings-count.

Another reason you may wish to modify schema generated by the API editor is to change the primitive data type (string, integer, float, etc.) of a schema element. JSON and XML primitive types do not map one-to-one with NonStop primitive types. When generating a schema 'by example', the API editor attempts to infer the corresponding NonStop data type and size of each example element's JSON or XML value. Sometimes the inference made by the API editor is incorrect – for example if the example data value is 1, the API editor infers a type of 'int' (32-bit), when the element, in fact, needs to hold values requiring a 64-bit integer. Again, you may edit the generated schema as needed to adjust any improperly inferred element data types.

Add Schema by Example for 'ExampleType'

```
{
  "ProductCategory": 123
}
```

Generated Schema for Example Above

```
  "ExampleType": {
    "anonymous": false,
    "elements": [
      {
        "name": "ProductCategory",
        "type": "int"
      }
    ]
  }
```

Because the example value of CustomerNumber provided is 123, the schema generator infers that the data type for the property should be 'int', which can store values between –2,147,483,648 to 2,147,483,647. However, if we know that product category will never be greater than 65,635, for example, we can modify "type" to be "unsignedShort". This will decrease the amount of space in the interprocess message (IPM) buffer from four bytes to two. Likewise, if we know that ProductCategory can be larger, we can modify "type" to be "unsignedLongLong", for example.

Schema Element Properties

The following properties may be applied to elements in the schema. Note that properties prefixed with *@Xml* are a subset of Java JAXB annotations.

property name	description
name	*required*; the name of the element
type	*required*; the type of the element; either a primitive type, or a type defined elsewhere in the schema. See below for supported primitive types.
ddlName	the name for the element that should be used in DDL generation; overrides the default name-to-ddlName mapping
dependsOn	when the element represents an array (minOccurs > 0), the name of another element in the current type definition that contains the actual count of 'in use' array elements
hide	boolean; specifies that the item should not be serialized, often used in an element targeted by dependsOn, sizeIs or scaleIs (synonymous with *private*)
isNull	boolean; when type is "nillable", the name of another element in the current type definition that indicates whether or not the field contains the value *null* when de/serialized
isSet	References an integer fields that indicates whether or not the field should be present in the request, or was present in the response. See Working with Optional Elements

property name	description
jsonType	the JSON type ("number", "string", or "boolean") that should be used to de/serialize the element; overrides the default primitive-type to JSON-type mapping
map	specifies that the element represents a JSON map
minOccurs	when "type" is "string", if minOccurs = 0, zero-length string values will not be serialized
maxOccurs	indicates the element represents an array; the maximum number of array elements that can be de/serialized
nillable	boolean; indicates that the field may contain the value *null*.
private	boolean; specifies that the item should not be serialized, often used in an element targeted by dependsOn, sizeIs or scaleIs. (synonymous with *hide*)
scale	for integer and numeric types, the scale that should be used in the generated DDL element
size	when "type" is "string", "base64Binary" or "hexBinary", the maximum length of the value
sizeIs	when "type" is "string", "base64Binary" or "hexBinary"; the name of another element in the current type definition that contains the actual length of the item de/serialized
stringPadding	"spaces" or "zeros"; indicates how LightWave expects to receive, returns string values from the client application. The default value is set in the API definition (which can be overridden by a CLIENT command-line parameter). Typically only set on the element level if more that one setting is needed for separate elements in a single API.
timestampFormat	when type="timestamp", indicates the de/serialization format: "ISO8601", "ISO8601:full-date", "RFC2822", or "XSD:dateTime"

<table>
<tr><th>property name</th><th>description</th></tr>
<tr><td>wrapped</td><td>boolean; indicates if an array is wrapped in an array container or a type/element is wrapped by its parent. Default is true.</td></tr>
<tr><td>@XmlAttribute</td><td>object; indicates that the value of the field is an attribute of the parent type. The object may contain the following properties
<table>
<tr><th>Property</th><th>Description</th></tr>
<tr><td>name</td><td>May be used to override the name of the attribute.</td></tr>
<tr><td>namespace</td><td>Indicates that the attribute it qualified with the specified namespace</td></tr>
</table>
<pre>
"anElement": {
  "anonymous": false,
  "elements": [
    {
      "name": "theAttribute",
      "type": "string",
      "size": 32,
      "@XmlAttribute": {}
    }
  ]
}
</pre>
Yields
<pre>
<anElement theAttribute="the-contents-of-theAttribute" />
</pre>
</td></tr>
</table>

<table>
<tr><th>property name</th><th>description</th></tr>
<tr><td>@XmlElement</td><td>object; allows various properties to be applied to an XML element. The object may contain the following properties:
<table>
<tr><th>Property</th><th>Description</th></tr>
<tr><td>name</td><td>May be used to override the name of the element.</td></tr>
<tr><td>namespace</td><td>Sets the namespace of the element. If omitted, the namespace is inherited from the parent element.</td></tr>
<tr><td>nillable</td><td>boolean; indicates that the field may contain the value null.</td></tr>
<tr><td>isNull</td><td>boolean; when type is "nillable", the name of another element in the current type definition that indicates whether or not the field contains the value null when de/serialized.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>property name</th><th>description</th></tr>
<tr><td>@XmlElementWrapper</td><td>object; causes a wrapper to be created around the element and is most commonly used to declare wrapped arrays. The object may contain the following properties:
<table>
<tr><th>Property</th><th>Description</th></tr>
<tr><td>name</td><td>May be used to override the name of the wrapper element.</td></tr>
<tr><td>namespace</td><td>Sets the namespace of the wrapper element. If omitted, the namespace is inherited from the parent element.</td></tr>
</table>
For example:
<pre>
{
  "RootElement": {
    "anonymous": false,
    "@XmlRootElement": {
      "name": "root"
    },
    "elements": [
      {
        "name": "theArray",
        "type": "string",
        "size": 32,
        "maxOccurs": 4,
        "@XmlElementWrapper": {
          "name": "aWrappedArray"
        },
        "@XmlElement": {
          "name": "item"
        }
      }
    ]
  }
}
</pre>
Yields
<pre>
<root>
  <aWrappedArray>
    <item>1</item>
</pre>
</td></tr>
</table>

<table>
<tr><th>property name</th><th>description</th></tr>
<tr><td></td><td>

```
    <item>2</item>
    <item>3</item>
  </aWrappedArray>
</root>
```

</td></tr>
<tr><td>@XmlRootElement</td><td>object; allows various properties to be applied to an XML document root element. The object may contain the following properties:
<table>
<tr><th>Property</th><th>Description</th></tr>
<tr><td>name</td><td>May be used to override the name of the element.</td></tr>
<tr><td>namespace</td><td>Sets the namespace of the root element. If omitted, the namespace is inherited from the parent element.</td></tr>
</table>
</td></tr>
</table>

property name	description

@XmlValue	object; causes the contents of the field to be mapped as the text content of the parent type. The object has no properties. For example:
	`"anElement": { "anonymous": false, "elements": [ { "name": "value", "type": "string", "size": 32, "@XmlValue": {} } ] }`
	Yields
	`<anElement>the-contents-of-value</anElement>`
	Not
	`<anElement> <value>the-contents-of-value</value> </anElement>`

Built-in 'primitive' Types

name	description	DDL type	JSON type
base64Binary	serialized as base-64 encoded string; n=size	PIC X(n)	string
boolean	zero serialized as "false"; non-zero values as "true"	BINARY 32	boolean
byte	-128 <= value <= 127	BINARY 8	number

name	description	DDL type	JSON type
double	double precision floating point	FLOAT 64	number
float	single precision floating point	FLOAT 32	number
hexBinary	serialized hex binary string; n=size	PIC X(n)	string
int	–2,147,483,648 to 2,147,483,647	BINARY 32	number
long	same as int	BINARY 32	number
longlong	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	BINARY 64	number
numeric	numeric string; n=precision; s=scale	PIC S9(n) PIC S9(n)V9(s)	number
numericSignEmbedded	numeric string; n=precision s=scale	PIC T9(n) PIC T9(n)V9(s)	number
numericSignTrailing	numeric string; n=precision s=scale	PIC 9(n)S PIC 9(n)V9(s)S	number
numericSignTrailingEmbedded	numeric string; n=precision s=scale	PIC 9(n)T PIC 9(n)V9(s)T	number
short	-32768 <= value <= 32767 (optional scale)	BINARY 16	number

name	description	DDL type	JSON type
string	character string s=size	PIC X(s)	string
timestamp	serialized as timestamp; see Working with Schema#timestampFormat	BINARY 64	string
unsignedByte	0 <= x <= 256	BINARY 8, UNSIGNED	number
unsignedInt	0 <= x <= 2^{32} - 1 (optional scale)	BINARY 32, UNSIGNED	number
unsignedLong	same as unsignedInt (optional scale)	BINARY 32, UNSIGNED	number
unsignedLongLong	0 <= x <= 2^{64} - 1 (optional scale)	BINARY 64, UNSIGNED	number
unsignedNumeric			number
unsignedShort	0 <= x <= 65535 (optional scale)	BINARY 16, UNSIGNED	number

Deploying APIs

An API must be deployed with the CLIENT process before it can be used by client applications.

To deploy an API:

1. Create the API definition using the LightWave Client Console.
2. Export the API definition from the LightWave Client filesystem into an API definition file through the Console or by using CUTILITY.

3. Start an instance of the CLIENT process using the API definition file. For more information see Starting the Client Process.

Generating IPM Definitions

Once you have deployed an API, you can create the interprocess message (IPM) definitions that your application will use to communicate with the CLIENT process and ultimately, the desired web service. Generating the IPM definitions is a two-step process:

- Create DDL Source
- Create IPM Definitions

Create DDL Source

Once an API is created you can create DDL definitions for the IPMs used to communicate with the CLIENT process.

To create DDL using the Console:

1. Select "APIs" from the console menu.
2. Select the desired API.
3. Click the icon to export the DDL.
4. Select whether you want to save the DDL to a local file or a Enscribe file. If Enscribe file is selected, supply the file name.
5. Click Save. The DDL source will be downloaded to your desktop or saved to the Enscribe file system.

To create DDL from the TACL command line using LWCCOM:

An API may be exported from the filesystem to DDL using the LWCCOM utility. The syntax of the export command is:

```
EXPORT API Command

Exports an API from the fileystem into a file

  EXPORT API <api-name>
         , FILE <file-name> [ ! ]
         [ , FORMAT { NATIVE | DDL } ]

<api-name>
```

```
  The name of the API.

FILE <file-name> [ ! ]

  File to receive exported API definition.

FORMAT { NATIVE | DDL }

  Export format: native API or DDL source. If omitted, NATIVE is assumed.
```

For example, to export API *the-api* to DDL file *theddl*, use the command:

```
tacl> run LWCCOM export api the-api, file theddl !, format ddl
```

The DDL creation process may change the names of some elements under certain conditions. The names will be changed in the following cases:

- **Web Service element names that are too long**
  DDL object names are limited to 30 characters. If while converting a Web service element name to a DDL name, the resulting name is longer than 26 characters, the name will be truncated.
- **Web service element name is a host language reserved word**
  If while converting a Web service element name to a DDL object name, the resulting name is the same as a host language reserved word, a '-rw' extension is appended to the name.
- **A child element has the same name as the parent element**
  If a child element of a web service type has the same name as the type itself, the child element name is altered by appending a 'Z' character to the name

You may control the DDL name of an element by using the *ddlName* annotation on the element definition. See Working with APIs.

Create IPM Definitions

The NonStop Data Definition Language (DDL) Compiler is a utility that is included with your NonStop system. It compiles DDL definitions into a binary format which can be subsequently output in the syntax of various programming languages such as C, COBOL, TAL or TACL. You use the DDL compiler to translate the DDL definitions into source code that you'll compile into your application.

```
tacl> DDL2
DDL Compiler T9100H01 - (01JAN17)  SYSTEM \NODE
!?DICT $MYVOL.MYSUBVOL
Dictionary opened on subvol $MYVOL.MYSUBVOL for update access.
!?COBOL mycopybk !
!?C myheader !
```

```
!?SOURCE ddldefs
!exit
```

This example opens a COBOL source file, "mycopybk", and a C source file, "myheader", for output. The "`?SOURCE`" command simultaneously loads and compiles the definitions from the "ddldefs" file (the output from CDF2DDL), translates the definitions to COBOL and C and stores the output in the appropriate files.

Notes for SCOBOL Users

Note that if you are generating a source file for use in SCOBOL programs, you should set the COBOL74 option before generating the output. An example of the use of this option is shown below:

```
> DDL2
DDL Compiler T9100H01 - (01JAN17)  SYSTEM \NODE
!?DICT $MYVOL.MYSUBVOL
Dictionary opened on subvol $MYVOL.MYSUBVOL for update access.
!?SETCOBOL74
!?COBOL mycopybk !
!?SOURCE ddldefs
!exit
```

Refer to the HPE *Data Definition Language (DDL) Reference Guide* for more information about the DDL Compiler.

Invoking API Requests

Once the CLIENT process is configured and running, any NonStop application can access the associated Web service by sending the process a predefined IPM request and waiting for the response. Each IPM request contains a LightWave header which contains a 2 byte method code. The method code indicates which Web service method should be invoked by this IPM request. The remainder of the IPM consists of fields that correspond to the parameters of the selected web service method.

For example, using the following fragment from the generated DDL for the sample Employees service API, to invoke a "Get Employees" request:

- Initialize the get-employees-rq structure to 0 values.
- Set the GET-EMPLOYEES-RQ.LIGHTWAVE-RQ-HEADER.RQ-CODE value to RQ-GET-EMPLOYEES.
- Optionally set the GET-EMPLOYEES-RQ.LIGHTWAVE-RQ-HEADER.RQ-TIMEOUT value to the number of milliseconds to wait for the request to complete.
- Set any remaining application element values.

- Send the IPM to the CLIENT process using the appropriate Guardian system procedure call.

> ⚠ To ensure compatibility with future versions of LightWave Client, the LIGHTWAVE-RQ-HEADER structure must be initialized to binary 0 before populating other values.

```
DEF EMPLOYEE-API-VAL.
  02 RQ-GET-EMPLOYEES                    PIC S9(4) COMP VALUE 1.
  02 RQ-ADD-EMPLOYEE                     PIC S9(4) COMP VALUE 3.
  02 RQ-ADD-EMPLOYEE-BLOB                PIC S9(4) COMP VALUE 6.
  02 RQ-ADD-EMP-BLOB-FILE                PIC S9(4) COMP VALUE 7.
  02 RQ-DELETE-EMPLOYEE                  PIC S9(4) COMP VALUE 4.
  02 RQ-GET-EMPLOYEE                     PIC S9(4) COMP VALUE 2.
  02 RQ-UPDATE-EMPLOYEE                  PIC S9(4) COMP VALUE 5.
END.

DEF LIGHTWAVE-RQ-HEADER.
  02 RQ-CODE                            TYPE BINARY 16.
  02 RQ-VERSION                         TYPE BINARY 16 UNSIGNED.
  02 RQ-TIMEOUT                         TYPE BINARY 32 UNSIGNED.
  02 RESERVED                           PIC X(24).
END.

!
! Operation:    GET /employees
! Description:  Get a list of employees
! Request Code: RQ-GET-EMPLOYEES
!
DEF GET-EMPLOYEES-RQ.
  02 LIGHTWAVE-RQ-HEADER                TYPE LIGHTWAVE-RQ-HEADER.
  02 BEGIN-AFTER                        PIC X(20).
END.
```

Once the CLIENT process invokes the Web service method it returns the response in an IPM. This response IPM also contains a standard CLIENT header which contains a 2 byte response code. If the response code is zero, the remainder of the IPM consists of fields that correspond to the output parameters of the Web service method. If the response code is non-zero then the application should treat the response as an error.

```
DEF LIGHTWAVE-RP-HEADER.
  02 RP-CODE                            TYPE BINARY 16.
  02 HTTP-STATUS                        TYPE BINARY 16.
  02 INFO-CODE                          TYPE BINARY 16.
  02 INFO-DETAIL                        TYPE BINARY 16.
  02 RESERVED                           PIC X(24).
```

```
END.

DEF GET-EMPLOYEES-200-RP.
  02 LIGHTWAVE-RP-HEADER                TYPE LIGHTWAVE-RP-HEADER.
  02 GET-EMPLOYEES-RESPONSE             TYPE GET-EMPLOYEES-RESPONSE.
END.
```

Error Handling

Every reply from the CLIENT process includes the LightWave reply header followed by user data if the request was successful or error information if an error occurred. The reply header has the following format:

```
DEF LIGHTWAVE-RP-HEADER.
  02 RP-CODE                            TYPE BINARY 16.
  02 HTTP-STATUS                        TYPE BINARY 16.
  02 INFO-CODE                          TYPE BINARY 16.
  02 INFO-DETAIL                        TYPE BINARY 16.
  02 RESERVED                           PIC X(24).
END.
```

The *rp-code* field will contain one of the following values:

```
DEF LIGHTWAVE-RP-CODE-ENUM TYPE ENUM BEGIN.
  89 LW-RP-SUCCESS                        VALUE 0.
  89 LW-RP-INFO                           VALUE 1.
  89 LW-RP-ERROR                          VALUE 2.
END.
```

These values should be interpreted as follows:

- If *RP-CODE* contains *LW-RP-SUCCESS*, the request was successful and user data follows the LightWave header.
- If *RP-CODE* contains *LW-RP-INFO*, the request was successful but string field or array truncation occurred while the response was being processed. In this case the contents of the *INFO-CODE* and *INFO-DETAIL* fields will describe the condition that occurred.
- If *RP-CODE* contains *LW-RP-ERROR* then an error occurred while processing the request.

Handling Field and Array Truncation Warnings

When *RP-CODE* contains *LW-RP-INFO*, the *INFO-CODE* field will contain one of the following values:

```
DEF LIGHTWAVE-INFO-CODE-ENUM TYPE ENUM BEGIN.
  89 LW-INFO-FIELD-TRUNCATED              VALUE 100.
  89 LW-INFO-ARRAY-TRUNCATED              VALUE 101.
END.
```

If *INFO-CODE* contains *LW-INFO-TRUNCATED* then a string value in the service response message was too long to copy to the associated string field in the reply IPM. In this case the *INFO-DETAIL* field contains the IPM offset of the string field that caused the warning.

If *INFO-CODE* contains *LW-INFO-OVERFLOW*, then an array in the response message contained more elements than were defined in the associated array in the reply IPM. In this case the *INFO-DETAIL* field contains the IPM offset of the array that caused the warning.

Although it is possible for multiple string or array truncations to occur in a single response, CLIENT only reports the first occurrence of the truncation.

Handling Errors

When *RP-CODE* contains *LW-RP-ERROR*, the reply IPM is formatted as a *LIGHTWAVE-ERROR-RP* structure which contains detailed information about the error.

```
DEF LIGHTWAVE-ERROR-RP.
  02 LIGHTWAVE-RP-HEADER                TYPE LIGHTWAVE-RP-HEADER.
  02 ERROR-SOURCE                       TYPE BINARY 32.
  02 ERROR-CODE                         TYPE BINARY 32.
  02 ERROR-SUBCODE                      TYPE BINARY 32.
  02 ERROR-MESSAGE                      PIC X(4096).
END.
```

The *ERROR-SOURCE* field contains one of the error source values and identifies the subsystem that reported the error.

```
DEF LIGHTWAVE-ERROR-SRC-ENUM TYPE ENUM BEGIN.
  89 LW-ERROR-SRC-LIGHTWAVE               VALUE 1.
  89 LW-ERROR-SRC-HTTP                    VALUE 2.
  89 LW-ERROR-SRC-TCPIP                   VALUE 3.
  89 LW-ERROR-SRC-SSL                     VALUE 4.
END.
```

If *ERROR-SOURCE* contains *LW-ERROR-SRC-LIGHTWAVE* then *ERROR-CODE* will contain one of the following *LIGHTWAVE-ERROR-ENUM* values:

```
DEF LIGHTWAVE-ERROR-ENUM TYPE ENUM BEGIN.
  89 LW-ERROR-INVALID-LICENSE              VALUE 1.
  89 LW-ERROR-INVALID-HEADER               VALUE 2.
  89 LW-ERROR-INVALID-RQ-CODE              VALUE 3.
  89 LW-ERROR-INVALID-TYPE                 VALUE 4.
  89 LW-ERROR-SERIALIZATION-ERROR          VALUE 5.
  89 LW-ERROR-DESERIALIZATION-ERROR        VALUE 6.
  89 LW-ERROR-REQUEST-TIMEOUT              VALUE 7.
  89 LW-ERROR-RESPONSE-NOT-DEFINED         VALUE 12.
  89 LW-ERROR-INVALID-URI-SCHEME           VALUE 100.
  89 LW-ERROR-INVALID-API-DEF              VALUE 101.
  89 LW-ERROR-INVALID-BASE-URI             VALUE 102.
  89 LW-ERROR-INVALID-HDR-VERSION          VALUE 103.
  89 LW-ERROR-SIGNATURE-ERROR              VALUE 104.
  89 LW-ERROR-INTERNAL-ERROR               VALUE 500.
END.
```

The *ERROR-SUBCODE* and *ERROR-MESSAGE* fields may contain additional information.

If *ERROR-SOURCE* contains any other error source value, the remaining error fields provide the error codes and message reported by that subsystem.

The error source value *LW-ERROR-SRC-SSL* may refer to an error encountered on an SSL or TLS connection.

The error source value *LW-ERROR-INVALID-TYPE* is not used and will not be returned. The value has been left in the definition in order avoid affecting existing applications that may reference it.

Advanced Topics

- Accessing Secure Web Services
- Using BLOBs
- Working with Numbers in Requests and Reponses
- Character String Encoding
- Sensitive Data Masking
- Working with Optional Elements
- Using MEASURE Counters
- Message Logging

- Working with API Parameters
- Working with Timestamps

Accessing Secure Web Services

LightWave Client provides several different mechanisms to allow you to access Web services securely. Which mechanism you must use, if any, is dictated by Web service.

- Using Credentials Files
- Request Authentication and Signing

Using Credentials Files

Some Web services may require clients to authenticate themselves before allowing access to the service. The authentication process uses credentials. There are two types of credentials supported by LightWave Client:

- a userid and password used for authentication
- a pass phrase associated with a PKCS12 certificate used for SSL client authentication

The type of authentication required, if any, is dependent on the Web service. If authentication is required, you must supply your credentials to the CLIENT process to use on your behalf .

LightWave Client provides a mechanism for storing your credentials in an encrypted format. Rather then specifying your credentials in clear text as a command line parameter whenever starting CLIENT, you can specify the name of an encrypted credentials file. The encrypted credential information can then be used by the CLIENT process for authentication when necessary.

Credentials files are created and validated using the LWCCOM utility. Refer to LWCCOM for more information. You can create as many credentials files as necessary, since different Web services may require different credentials.

This example stores the userid 'myuserid' and password 'mypassword' in the encrypted credentials file "mylogin", then supplies the credentials filename with the -httpauth command line parameter when starting SOAPAMCP.

```
tacl> run lwccom create credentials mylogin !, credentials myuserid:mypassword
CREDENTIALS file $VOL.SUBVOL.MYLOGIN created
tacl> run client / name $cli / -http-credentials +mylogin @otheropts
```

This example stores the pass phrase in the encrypted credentials file "certpass". Since the credentials parameter was not specified, LWCCOM prompts the user for the credentials without echoing them to the terminal.

```
tacl> run lwccom create credentials certpass !
Enter credentials:
```

```
Re-enter credentials:
CREDENTIALS file $VOL.SUBVOL.CERTPASS created
tacl> run client / name $cli / -client-cert mycert +certpass @otheropts
```

Note that the credentials file is location dependent, i.e., if the file is moved or renamed, the credentials cannot be decrypted. This feature is designed to disable the credentials in case the file is stolen or otherwise used inappropriately. You can specify an alternate target location for the credentials file if the file is to be created in one location but moved to another location for deployement. This example creates a credentials file on the current volume for eventual deployment on the $SYSTEM volume.

```
tacl> run lwccom create credentials mylogin !, credentials myuserid:mypassword,
target $system.deploy.mylogin
CREDENTIALS file $VOL.SUBVOL.MYLOGIN created
```

Even though credentials become invalid when moved, system administrators should still use Guardian security to prevent unauthorized access to the contents of the file.

This example verifies the contents of the credentials file.

```
tacl> run lwccom validate credentials mylogin
CREDENTIALS file $VOL.SUBVOL.MYLOGIN is valid
```

Request Authentication and Signing

Some cloud based services require requests to be *signed* using service specific signature algorithms. The CLIENT process supports automatic request signing for a number of services. To enable request signing, the application developer configures an authentication configuration file which defines the configuration parameters. Configuration parameters may also be supplied in the request itself and may be configured in the API Editor.

The Authentication Configuration File

The authentication config file, or *auth config*, uses the same format described in the section Configuration File Format. The section name is *auth.* The auth section always contains the method param indicating the auth method, followed by method specific auth params. The location of the configuration file is then supplied to the CLIENT process using the *--auth* startup option. Changes to the auth config file may be monitored using the *--monitor* option.

An example auth config file for Amazon Web Services Signature Version 4 is shown below:

```
[auth]
Method=aws-signature-v4
AccessKeyId=AKID8D8WDF88E8F8E8FA
```

```
SecretAccessKey=yavFGsdfjoielskdfjieaiehihjoiij5Dfq9i5qD
```

An example of the startup options required to use the auth config file is shown below:

```
tacl>run CLIENT --api $vol.subvol.awsapi &
--base-url https://s3.us-east-1.amazonaws.com &
--log $vol.subvol.awslog &
--auth $vol.subvol.awsauth &
--monitor auth:30
```

Service Specific Auth File Options

The following signing methods are currently supported with new methods to be added in future releases. If your application requires a signing method not listed here, please visit the Support Center and open a ticket.

Note that parameter names in the auth config are *not* case sensitive.

- Amazon Web Services Signature V4
- Azure Event Hub
- Azure IoT Hub
- CyberSource HTTP Signature Authentication
- Generic Authorization Header with Token

Amazon Web Services Signature V4

This method can be used with any AWS service that requires signature version 4.

Param Name	Param Value
method	aws-signature-v4
AccessKeyId	An AWS access key ID
SecretAccessKey	The secret access key associated with the Access Key ID
SessionToken	The session token associated with the AWS access key ID. This param is optional, but required if the AccessKeyId and SecretAccessKey were obtained from the AWS Security Token Service (STS). If omitted, no SessionToken value is used.

Param Name	Param Value
Region	The AWS region for the request. This param is optional and if omitted, is derived from the base URL.
Service	The AWS service for the request. This param is optional and if omitted, is derived from the base URL.
ClockCorrection	In the event that the system time is incorrect, this param may be used to adjust the current time used by the signature algorithm. The value is specified in number of seconds, + or -, to adjust the time used by the signature algorithm to the correct time. This param is optional and if omitted, the current system time is used. This option should only be used in exceptional cases in which the system clock cannot be synchronized to real world time.

Azure Event Hub

This method can be used with the Azure Event Hub

Param Name	Param Value
method	azure-event-hub
SharedAccessKeyName	The shared access key name of an appropriate access policy
SecretAccessKey	The secret access key of the access policy.
TokenTTL	The length of time the token should be valid, in seconds. This param is optional and if omitted, defauts to 60 seconds.
ResourceURI	The resource URI to use when creating the Azure signature. This parameter is optional and if omitted, is derived from the request URL.

Param Name	Param Value
TokenExpiry	An explicit token expiry time expressed in seconds since Unix Epoch (Jan 1, 1970). If specified, the tokenTTL param is ignored. If omitted, TokenTTL is used to calculate the token expiration time. This param should be used for testing purposes only.
ClockCorrection	In the event that the system time is incorrect, this param may be used to adjust the current time used by the signature algorithm. The value is specified in number of seconds, + or -, to adjust the time used by the signature algorithm to the correct time. This param is optional and if omitted, the current system time is used. This option should only be used in exceptional cases in which the system clock cannot be synchronized to real world time.

Azure IoT Hub

This method can be used with the Azure IoT Hub

Param Name	Param Value
method	azure-iot-hub
SharedAccessKeyName	The shared access key name of an appropriate access policy
SecretAccessKey	The secret access key of the access policy.
TokenTTL	The length of time the token should be valid, in seconds. This param is optional and if omitted, defauts to 60 seconds.
ResourceURI	The resource URI to use when creating the Azure signature. This parameter is optional and if omitted, is derived from the request URL.

Param Name	Param Value
TokenExpiry	An explicit token expiry time expressed in seconds since Unix Epoch (Jan 1, 1970). If specified, the tokenTTL param is ignored. If omitted, TokenTTL is used to calculate the token expiration time. This param should be used for testing purposes only.
ClockCorrection	In the event that the system time is incorrect, this param may be used to adjust the current time used by the signature algorithm. The value is specified in number of seconds, + or -, to adjust the time used by the signature algorithm to the correct time. This param is optional and if omitted, the current system time is used. This option should only be used in exceptional cases in which the system clock cannot be synchronized to real world time.

CyberSource HTTP Signature Authentication

This method can be used with the CyberSource REST API.

Param Name	Param Value
method	cybersource-http
MerchantId	The CyberSource Merchant ID.
MerchantKeyID	A CyberSource API key ID.
MerchantSecretKey	The shared secret key associated with the MerchantKeyID.

Generic Authorization Header with Token

This method allows an arbitray HTTP header name, value pair to be supplied as an authentication method. This method can be used for custom authentication methods or for standard authentication methods such as OAuth 2 Bearer tokens.

Param Name	Param Value
method	auth-token
Header	The name of the HTTP header.
Token	The value of the HTTP header.

Using BLOBs

There may be cases where an application requires more control over the Web service request or response payload. For example, if the Web service requires a message structure that can't be created or parsed by the native LightWave Client JSON support, or if the message payload is larger than the maximum message size allowed by an interprocess message. For cases like these, LightWave Client 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 the HTTP Response Body Source or Target Content Schema as BLOB, and the Data Source or Target as IPM, the DDL generated for the operation will contain a single character field representing the data that is to be sent with the request or that was received in the response. The BLOB-SIZE field indicates the actual size of the data to send or the size of the data that was received. The data in the BLOB field represents exactly the data that is to be sent or that was received, it is not manipulated by the CLIENT process in any way. The maximum size for an IPM BLOB is 2097116.

```
DEF ADD-EMPLOYEE-BLOB-RQ.
  02 LIGHTWAVE-RQ-HEADER                TYPE LIGHTWAVE-RQ-HEADER.
  02 BLOB-SIZE                          TYPE BINARY 32 UNSIGNED.
  02 BLOB                               PIC X(4096).
END.

DEF ADD-EMPLOYEE-BLOB-201-RP.
  02 LIGHTWAVE-RP-HEADER                TYPE LIGHTWAVE-RP-HEADER.
  02 BLOB-SIZE                          TYPE BINARY 32 UNSIGNED.
  02 BLOB                               PIC X(4096).
END.
```

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 HTTP Response Body Source or Target Content Schema as BLOB, and the Data Source or Target as FILE, the DDL generated for the operation will contain a single character field representing the name of the Enscribe file representing the data that is to be sent with the request or that was received in the response. The data in the file BLOB represents exactly the data that is to be sent or that was received, it is not manipulated by the CLIENT process in any way. File BLOBs are Enscribe unstructured files that are treated as binary streams by the CLIENT process.

Request BLOB files are created by the application and their location is provided in the request IPM. Note that EDIT files (type 101) are read as text files and will be transferred with each line ending in a single newline (0x0A) character. All other file types are transferred as binary streams.

Response BLOB files are created by the CLIENT process as type 180 files (binary stream) in the location specified by the CLIENT process *--blob-files* startup option. The name of the file is returned in the response IPM. Note that the application is responsible for the disposal of the request and response BLOB files. See CLIENT for more information on the *--blob-files* startup option.

```
DEF ADD-EMP-BLOB-FILE-RQ.
  02 LIGHTWAVE-RQ-HEADER                TYPE LIGHTWAVE-RQ-HEADER.
  02 BLOB-FILE-NAME                     PIC X(256).
END.

DEF ADD-EMP-BLOB-FILE-201-RP.
  02 LIGHTWAVE-RP-HEADER                TYPE LIGHTWAVE-RP-HEADER.
  02 BLOB-FILE-NAME                     PIC X(256).
END.
```

Working with Numbers in Requests and Reponses

LightWave exchanges messages with REST web services in JSON, or Javascript Object Notation, format. Like Javascript, JSON does not support an integer data type, only a Number type that is defined as a double-precision 64-bit binary format IEEE 754 value (number between $-(2^{53}-1)$ and $2^{53}-1$).

LightWave relies on schema – definitions of the structure and type of the JSON elements that make up request and responses – in order to translate data to and from structured interprocess messages that client applications can easily understand. At design-time, LightWave can automatically create request and response schemas from sample JSON requests and responses that you supply (typically documented by the owner of the web API you're working with). These schemas ultimately determine the structure of the request and response IPMs (interprocess messages) that your client application will use to invoke

web APIs through LightWave at runtime. When the sample requests and responses contain numeric values, LightWave generates schema that represents the values as *integer* data types (including scaled integers) when possible. This behavior is based on the experience that numeric values in web APIs typically represent integral or fixed-point decimal values, not floating point values, and that COBOL programs cannot easily process floating point numbers.

For example, the consider the following JSON sample data represents a response returned from a fictional web API:

```
{
  "price":  12345.67,
  "quantity":  987,
  "pi":  3.14159265358979323846264338327950288,
  "avagadro": 6.02E+23
}
```

LightWave's 'schema by example' function generates the following schema for the sample above:

```
"my-sample": {
  "elements": [
    {
      "name": "price",
      "type": "int",
      "scale": 2
    },
    {
      "name": "quantity",
      "type": "int"
    },
    {
      "name": "pi",
      "type": "longlong",
      "scale": 15
    },
    {
      "name": "avagadro",
      "type": "double"
    }
  ]
}
```

Based on the sample value of 12345.67, LightWave generates a schema type for "price" of 'scaled 32-bit signed integer' (also known as BINARY 32,2 or S9(7)V99 COMP). Based on your knowledge of the range of data that could be returned by the API, you may decide to change the type and/or scale of the schema for "price". For example, you could change it

to "type": "longlong", "scale": 3 to support values with up to 15 digits to the left of the decimal and three to the right. Or, you could change type to "float" and remove "scale", indicating that you do, in fact, want to use the value as a floating point number in your application.

In the second example value, since 987 is an integral value, schema type "int" is generated. Again, you may modify the type and add "scale" if appropriate. In the third example, even though the value of pi is specified to 33 digits, Javascript truncates the value to the approximate 3.141592653589793, since this is the highest precision it can store. In this case LightWave chooses a 64-bit "longlong" with scale = 15 for the schema. The final example shows that LightWave generates a schema type of "double" for Avagradro's Number, since the value cannot be represented as a scaled integer.

You may modify the schema that LIghtWave generates based on the examples you provided. However, after doing so, you must re-export DDL definitions for your API, then use the NonStop DDL compiler to regenerate language-specific (C/C++, COBOL, etc.) header files that correspond to the new schema. Finally, it may also be necessary to adjust your client application source code to account for the schema changes.

Here's the DDL that's generated from the schema above.

```
DEF MY-SAMPLE.
  02 PRICE                              TYPE BINARY 32,2.
  02 QUANTITY                           TYPE BINARY 32.
  02 PI                                 TYPE BINARY 64,15.
  02 AVAGADRO                           TYPE FLOAT 64.
END.
```

C/C++ Programming Considerations

Using the DDL source above and the NonStop DDL compiler, the following C/C++ header definition is created:

```
#include <tnsint.h>
typedef struct __my_sample
{
   __int32_t                        price;
   __int32_t                        quantity;
   long long                        pi;
   double                           avagadro;
} my_sample_def;
```

However, the following warnings are issued due to the fact the scaled integers are not natively supported in C/C++.

```
    *** WARNING *** C OUTPUT DIAGNOSTICS:
    *** WARNING *** Unsupported data type in element PRICE
    *** WARNING *** Unsupported data type in element PI
```

Your client application *can* still manipulate scaled integers as shown below:

```
my_sample_def  sample;

/* constructing values */
int dollars;
int cents;
sample.price = dollars * 100 + cents;

int whole = 3;
longlong fraction = 141592653589793;
sample.avagadro = whole * 1000000000000000 + fraction;

/* deconstructing values */
dollars = sample.price / 100;
cents = sample.price % 100;

whole = (int)( sample.avagadro / 1000000000000000 );
fraction = sample.avagadro % 1000000000000000;
```

You can also use Guardian System Procedures FORMATCONVERTX / FORMATDATAX to handle input/output for scaled integers in C programs.

COBOL Programming Considerations

Using the DDL source above and the NonStop DDL compiler, the following COBOL definition is created:

```
01 MY-SAMPLE.
  02 PRICE                        PIC S9(7)V9(2) COMP.
  02 QUANTITY                     NATIVE-4.
  02 PI                           PIC S9(3)V9(15) COMP.
  02 AVAGADRO                     PIC X(8).
```

COBOL supports scaled integers natively. However, the following warnings are issued due to the fact that floating point values are not supported in COBOL.

```
    *** WARNING *** COBOL 85 OUTPUT DIAGNOSTICS:
```

```
    *** WARNING *** Unsupported data type in element AVAGADRO
```

You can use Guardian System Procedures FORMATCONVERTX / FORMATDATAX to handle input/output for floating point numbers in COBOL programs.

Character String Encoding

When LightWave Client copies "string" fields between the JSON message elements and the application IPM it assumes that the data is stored in the IPM using the default encoding which is ISO-8859-1 encoding. The encoding used can be altered on the element, the API, or set at CLIENT 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 CLIENT instance set the --default-encoding startup option to the desired encoding.

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

```
"Request": {
  "elements": [
    {
      "name": "string_SHIFT_JIS",
      "type": "string"
      "size": 32,
      "encoding": "SHIFT_JIS"
    }
  ]
}
```

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

```
{
  "string_SHIFT_JIS": "ぐけげこござ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 Client always uses UTF-8

encoding for the message payload. Refer to Character Encoding Names for a complete list of the available character encoding names.

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 diagnostic and HTTP logs and in memory. For example, a request or response containing the following schema structure:

```
"CardHolderData": {
  "elements": [
    {
      "name": "name",
      "type": "string"
      "size": 32
    },
    {
      "name": "cardNumber",
      "type": "string",
      "size": 32,
      "sensitive": true
    }
  ]
}
```

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 Client has several mechanisms to indicate the presence of optional elements. These mechanisms use a schema property to indicate the presence or absence of the element.

- Using the hideIfEmpty Property
- Using the isSet Property
- Using the minOccurs Property

Using the hideIfEmpty Property

The *hideIfEmpty: true* property may be set on any element to indicate that if the source element is *empty*, it should not be included in the request 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 property, 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 request payload.

```
{
  "SampleType": {
    "elements": [
      {
        "name": "optionalIfSpaces",
        "type": "string",
        "size": 32,
        "hideIfEmpty": true,
        "stringPadding": "spaces"
      },
      {
        "name": "optionalIfZeros",
        "type": "string",
        "size": 32,
        "hideIfEmpty": true,
        "stringPadding": "zeros"
      }
    ]
  }
}
```

Using the isSet Property

The *isSet* property may be set on any element, and references an integer element which is used to indicate the presence or absence of the source element. In the following example,

element theAddress is included in the request payload only if the value of element *isAddressSet* is nonzero. In this example, the *hide* property is set on the *isAddressSet* element to indicate that it is never included in the request payload. Note that when a response is received, the *isSet* target element is set to zero if the source element is not present in the response, or nonzero if it is present.

```
{
  "SampleType": {
    "elements": [
      {
        "name": "isAddressSet",
        "type": "int",
        "size": 4,
        "hide": true
      },
      {
        "name": "theAddress",
        "type": "AddressType",
        "isSet": "isAddressSet"
      }
    ]
  }
}
```

Using the minOccurs Property

Using the minOccurs property as a mechanism to hide string elements is deprecated. New definitions should use *hideIfEmpty*.

The *"minOccurs" : 0* property 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 request 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 request payload

```
{
  "SampleType": {
    "elements": [
      {
        "name": "optionalIfSpaces",
        "type": "string",
        "size" : 32,
        "minOccurs": 0,
```

```
        "stringPadding": "spaces"
      },
      {
        "name": "optionalIfZeros",
        "type": "string",
        "size": 32,
        "minOccurs": 0,
        "stringPadding": "zeros"
      }
    ]
  }
}
```

Using MEASURE Counters

LightWave Client supports several 'user defined' MEASURE counters which you can use to help analyze performance of your LightWave Client application. Each counter is an array counter in which each element of the array corresponds to an operation within the API that LightWave Client is currently supporting. For example, the nth index of the LWC-CALL-COUNT counter corresponds to the nth index of the API operation array. To display the API operations and their indexes, use the LWCCOM utility.

The available MEASURE counter names and types are shown below.

Counter Name	Type	Description
lwc-call-count	accum,array	The number of API calls.
lwc-connect	qbusy, array	The amount of time spent connecting to the remote API endpoint.
lwc-serialize	qbusy, array	The amount of time spent serializing the API request (converting the structured IPM to JSON format).
lwc-sign	qbusy, array	The amount of time spent signing the API request, when required.
lwc-request	qbusy, array	The amount of time spent sending the request to the remote API endpoint.

Counter Name	Type	Description
lwc-response	qbusy, array	The amount of time spent receiving the response from the remote API endpoint.
lwc-deserialize	qbusy, array	The amount of time spent de-serializing the API response (converting the JSON formatted message to the structured IPM format).
lwc-diag-log	qbusy, array	The amount of time spent generating the diagnostic log.
lwc-msg-log	qbusy, array	The amount of time spent filtering and delivering the message logging events.
lwc-total	qbusy, array	The total amount of time spent processing the request and response. This value may not exactly match the sum of the other busy counters due to some miscellaneous processing not measured by the other counters. Note that the total time does not account for time spent waiting for the remote API service endpoint to respond.

When adding counters to your measurement, be sure to specify arrays that are at least as large as the number of operations (paths/methods) in the API, otherwise some operations will not be measured.

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

+ add userdef CLIENT

== where ARRAY 10, for example, represents the number of
== operations in the API in use; each one has its own count

+ add counter lwc-call-count, process client, accum, array 10
+ add counter lwc-connect, process client, qbusy, array 10
+ add counter lwc-serialize, process client, qbusy, array 10
+ add counter lwc-sign, process client, qbusy, array 10
+ add counter lwc-request, process client, qbusy, array 10
+ add counter lwc-response, process client, qbusy, array 10
+ add counter lwc-deserialize, process client, qbusy, array 10
```

```
+ add counter lwc-diag-log, process client, qbusy, array 10
+ add counter lwc-msg-log, process client, qbusy, array 10
+ add counter lwc-total, process client, qbusy, array 10

+ start measurement LWCMEAS
== run your application
+ stop measurement LWCMEAS

+ 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 Client™ CLIENT process 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 forwarded to a specified collector target process as an inter-process message. Configuration details for the collector process and the desired request information is provided in a CLIENT process Configuration File.

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

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

Hereinafter, the *sending process* refers to the CLIENT process.

The Message Log Request IPM

The sending process forwards 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
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.

Data Type	Description
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*.
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.
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	TCP/IP connection time. May be 0 if a cached connection was used.

Field Name	Data Type	Description
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.
SERVER-IO-TIME	Interval	Always 0.
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".

Field Name	Data Type	Description
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 Request User Data**.
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. The collector process should issue a sufficient number of $RECEIVE read operations in order to read the entire request. Pathway process collectors must use Pathway dialogs to process the request stream.

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.

```
{
  "RequestMessage": {
    "anonymous": false,
    "elements": [
      {
        "name": "requestData",
        "type": "string",
        "size": 32
      },
      {
        "name": "requestId",
        "type": "string",
        "size": 32,
        "msgLogUserData" : true
      }
    ]
  }
}
```

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.

Name	Value
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.
process-ancestor-program	The program file name of the sending process ancestor.

Name	Value
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 disabled by starting the sending 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 CLIENT --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 LWCCOM CHECK command.

```
tacl> run lwccom
LightWave Client COM 1.2.0 - \NODE.$X123
Copyright (c) 2020 NuWave Technologies, Inc. All rights reserved.
LWCCOM 1-> check config mlogcfg
Checking CONFIG file MLOGCFG.
The file contains a valid 'msg-log' configuration.
The CONFIG file is valid.
LWCCOM 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.

The examples use the *match-all* element, but all options may also be used with *match-any*.

Option	Description

alias	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*"`

http-status	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"`

method	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"`

Option	Description

path	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/*"`

reply-code	A single range or sequence of ranges used to select or ignore reply-code values. See Using Patterns. May be negated, see Using Negation.
	Example
	`filters:`
	`  match-when-0:`
	`  match-all:`
	`  reply-code: "0"`
	`  match-when-10-20-but-not-15:`
	`  match-all:`
	`  reply-code:`
	`  - "!15"`
	`  - "10:20"`

Option	Description

rq-header / rp-header	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.
	Boolean true of false may also be specified to match when the header is or is not present.
	Example
	`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`

total-time-exceeds	A microsecond value used to select or ignore a message when the TOTAL-TIME value (request time in microseconds) exceeds the value. May be negated, see Using Negation.
	Example
	`filters:`
	`  match-when-less-than-2-seconds:`
	`  match-all:`
	`  total-time-exceeds: "!20000000"`
	`  match-when-exceeds-2-seconds:`
	`  match-all:`
	`  total-time-exceeds: "20000000"`

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 collector processes in C and COBOL can be found on Github.

Working with API Parameters

API Parameters allow data mapping values in an API definition to be supplied as CLIENT process startup options. Possible use cases for this are:

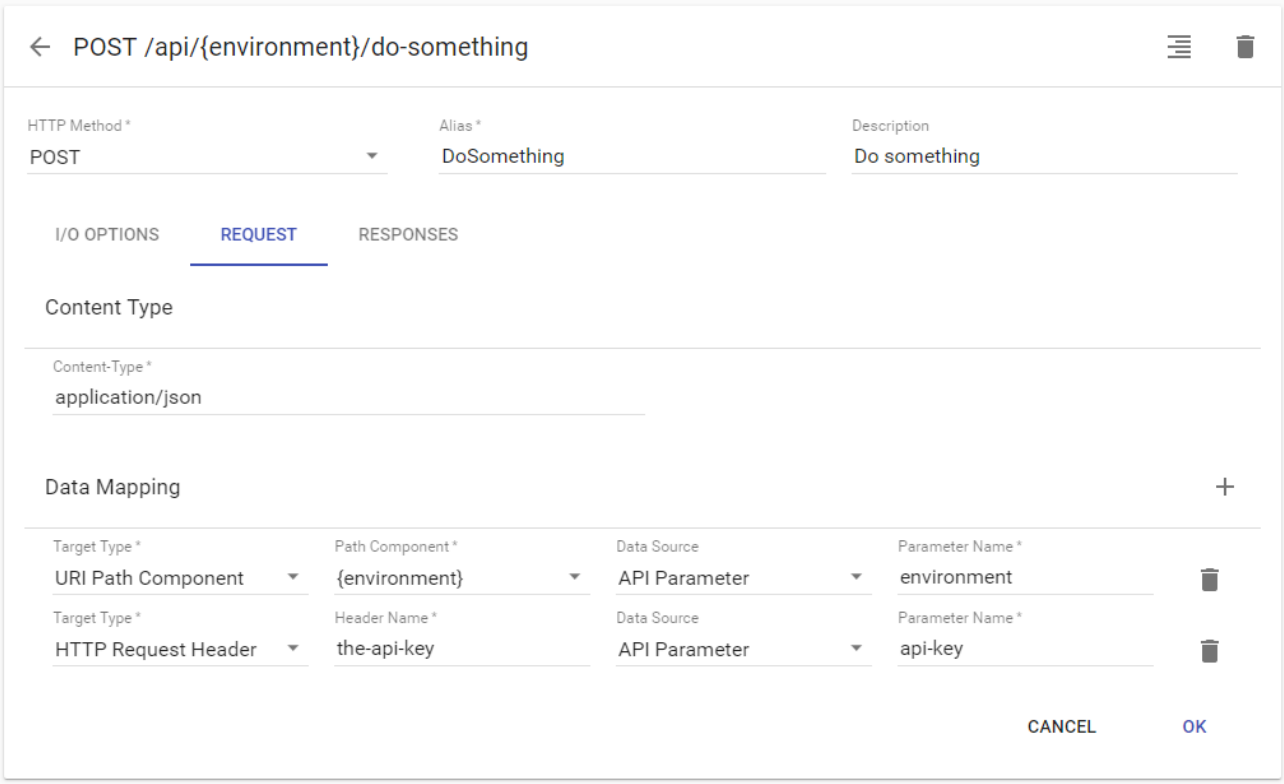
- A request URI is */api/**dev**/do-something* in the development environment but is */api/**prod**/do-something* in the certification / production environment. API parameters can be used to alter the URI with a value from the CLIENT process startup configuration, eliminating the need to maintain two versions of the API definition.

- A service requires an API key sent in an HTTP header. The value of the API key can be supplied in the CLIENT process configuration, instead of by the application.

The following example illustrates how to alter both the request URI and an API key, without changing the API definition, when migrating an API from a development environment to a certification environment. The steps are:

- Create an operation path with a path component which will contain the environment. For example: */api/**{environment}**/do-something*
- Create the operation. On the Request tab, in the Data Mapping section, add these mappings:
  - Target Type *URI Path Component*, Path Component *{environment}*, Data Source *API Parameter*, Parameter Name *environment*.
  - Target Type *HTTP Request Header*, Header Name *the-api-key*, Data Source *API Parameter*, Parameter Name *api-key*.
- In the CLIENT process configuration, add options for *--api-param-environment* and *--api-param-api-key* with the appropriate values.

API Definition



Development Serverclass

```
== Settings for linkdepth, maxservers, maxlinks, etc are examples, not
recommendations.
reset server
set server cpus           0:1
set server createdelay    0 secs
set server deletedelay    15 secs
set server highpin        on
set server linkdepth      5
set server maxservers     1
set server maxlinks       20
set server numstatic      0
set server program        client
set server tmf            off
set server debug          off
set server param          api "apifile"
set server param          base-url "https://dev.example.com"
set server param          diag-log "+logconf"
set server param          log "+logconf"
set server param          monitor "api:5 diag-log:5 log:5"
set server param          tcpip-process "$ztc0"
set server param          api-param-environment "dev"
set server param          api-param-api-key "dev-api-key"
add server example
```

Certification Serverclass

```
== Settings for linkdepth, maxservers, maxlinks, etc are examples, not
recommendations.
reset server
set server cpus           0:1
set server createdelay    0 secs
set server deletedelay    15 secs
set server highpin        on
set server linkdepth      5
set server maxservers     1
set server maxlinks       20
set server numstatic      0
set server program        client
set server tmf            off
set server debug          off
set server param          api "apifile"
set server param          base-url "https://cert.example.com"
set server param          diag-log "+logconf"
```

```
set server param          log "+logconf"
set server param          monitor "api:5 diag-log:5 log:5"
set server param          tcpip-process "$ztc0"
set server param          api-param-environment "prod"
set server param          api-param-api-key "prod-api-key"
add server example
```

Working with Timestamps

LightWave Client 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 property to indicate if the field is 64 bit or 48 bit, for example:

```
{
  "TimestampTypes": {
    "anonymous": false,
    "elements": [
      {
        "name": "timestamp64bitJulian",
        "type": "timestamp",
        "size": 8
      },
      {
        "name": "timestamp48bit",
        "type": "timestamp",
        "size": 6
      }
    ]
  }
}
```

Timestamp Formats

Several standard timestamp formats can be serialized or deserialized from the message body. The timestamp format is specified using the *timestampFormat* property. 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.

timestampFormat	description
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

```
{
  "ISO8601Timestamp": {
    "anonymous": false,
    "elements": [
      {
        "name": "theDate",
        "type": "timestamp",
        "size": 8,
        "timestampFormat": "ISO8601:full-date"
      }
    ]
  }
}
```

Troubleshooting

Use the following LightWave Client tools to troubleshoot the interaction between LightWave Client and the target web 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 Client instance. Diagnostic logging should not be enabled on production instances unless necessary.

To enable diagnostic logging for an API, use the --diag-log option. For more information see CLIENT command line reference and Diagnostic Log Configuration. Diagnostic logs may be viewed as Enscribe files or from the Console. For more information on viewing logs from the Console and log content see Diagnostic Logs.

Character Encoding Names

LightWave Client 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 Client 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 Client

These instructions apply to new installations. See Updating LightWave Client 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:

- CONSOLE- The LightWave Client Console process.
- CLIENT - The LightWave Client CLIENT process.
- CUTILITY - The LightWave Client Utility process.
- ICUDATA - Character set translation data.
- CCPnnnnn - The console installation package (where nnnnn is a version number)
- CAROOT - A bundle of root certificate authority certificates
- RWORDS - A list of C and COBOL reserved words that are installed for use by the DDL generator.
- ZLOGOPTS - Sample logging options file
- ZSTARTPW - Sample CLIENT process Pathway configuration startup macro
- ZSTARTUP - Sample CONSOLE process startup macro
- LWMLDDL - Message Logging DDL.

> ⚠ CUTILITY has been deprecated and will no longer be updated. LWCCOM has replaced CUTILITY as the LightWave Client management CLI.

Overview of Installation

The installation process includes the following steps:

- Obtain a Product License
- Download the Release Package
- Install the Release Package
- Create a Filesystem
- Optionally Install the Console Package
- Install the Product License
- Start the Console Process

Obtain a Product License

A license is required to run LightWave Client. 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 Client 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 Client stores its operational data in a filesystem consisting of a collection of Enscribe files. A filesystem is created using the LWCCOM application as follows:

```
> run LWCCOM 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 Client Console provides a browser-based management interface to LightWave Client. 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 LWCCOM program. In order to use the Console, install the console package as follows:

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

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

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 Client installation subvolume
- the file LICLWC in the LightWave Client installation subvolume
- the file $SYSTEM.NUWAVE.LICLWC

The LightWave Client CLIENT 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 Console Process

The LightWave Client Console process supports the user interface for the product. Refer to Starting the Console Process for further instructions.

Once the Console 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 Console process. Login to the LightWave Client Console application using the username 'admin' and the password 'password' (or the new password you specified when you created the filesystem).

Updating LightWave Client

The process for updating LightWave Client 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 have 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:

- CONSOLE- The LightWave Client Console process
- CLIENT - The LightWave Client CLIENT process
- LWCCOM - The LightWave Client Management CLI
- ICUDATA - Character set translation data.
- CCPnnnnn - The console installation package (where nnnnn is a version number)
- CAROOT - A bundle of root certificate authority certificates
- RWORDS - A list of C and COBOL reserved words that are installed for use by the DDL generator
- ZLOGOPTS - Sample logging options file
- ZSTARTPW - Sample CLIENT process Pathway configuration startup macro
- ZSTARTUP - Sample CONSOLE process startup macro
- CUTILITY - The LightWave Client Utility process

> CUTILITY has been deprecated and will no longer be updated. LWCCOM has replaced CUTILITY as the LightWave Client management CLI.

Overview of Update Installation

The update process includes the following 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 Client. From the existing LightWave Client installation subvolume, use the FILEINFO command to ensure that no files are open. Stop any processes that have files opened. From the LightWave Client installation subvolume:

```
> status *, prog CLIENT, stop
> status *, prog CONSOLE, stop
```

Perform a Backup of the Existing Installation

Verify that you have a current backup of the existing installation, including the LightWave Client filesystem (FC0* 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 Client filesystems (FC0* 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

The LightWave Client filesystem contains information necessary to support the LightWave Client Console application, including API definitions you have created and the Console web application itself. The filesystem must be upgraded to incorporate any structural or content modifications included in the new release. The filesystem is updated using the LWCCOM utility, which is run as follows:

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

where <filesystem-subvolume> is the name of the subvolume that contains the LightWave Client filesystem (FC0* files).

Optionally Install the Updated Console Package

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

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

Restart Processes

Restart the LightWave Client components (CONSOLE, CLIENT) as per your locally-developed procedures.

Starting LightWave Client

LightWave Client consists of two major component processes: Console and Client.

- Starting the Console Process
- Starting the Client Process

Console

The role of the Console process is to support the administrative and management user interface for LightWave Client. Console provides a browser interface which allows you to:

- View system status dashboard
- Manage APIs
- Manage Access Control Policies
- Manage Services
- View HTTP and diagnostic logs
- Manage SSL/TLS Client, Server, and Root CA Certificates
- Manage LightWave Client users and groups

Once the Console process has been started, you access the LightWave Console application by entering the following URL in your web browser.

http://*system-name*:*port-number*

where system-name is your NonStop server's DNS name or IP address, and port-number is the console port number that was assigned when the Console process was started. If you

configure the Console process with an Server Certificate, you can access it via https using the appropriate port.

The Console process uses the LightWave File System for its data store, which is shared with the Client process. The Console process is not required for the runtime operation of LightWave Client services. You may choose to shut down Console on production systems after initial configuration has been completed as an additional security measure.

Client

The role of Client is to provide the runtime implementation of a LightWave Client API you defined earlier using Console. Although you can run one or more client processes for a given API (e.g., for load balancing), each instance of the Client process supports exactly one API. The Client process accepts an interprocess message on $RECEIVE from a client application, which it transforms into a REST web-API request and forwards to a service endpoint using HTTP over TCP/IP. When the service responds, Client transforms the response into an interprocess message which it REPLYs to the original client requester application. Each Client process is capable of supporting simultaneous requests from multiple client applications.

Starting the Console Process

The LightWave Client Console process runs as a single fault tolerant process pair and may be configured to listen on one or more TCP/IP processes and ports.

> **Tip**
>
> A sample LightWave Client Console process 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 the Process

The Console is started by running the CONSOLE program from TACL.

```
> run console / run-options / command-line-options
```

Examples

Start the Console 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. This example assumes that a server certificate with Common Name lightwave.example.com has already been installed.

```
> run console / name $LWCON, cpu 0 / --backupcpu 1 --console-ports $ztc0:8080
$ztc0:8443:lightwave.example.com
```

More Information

For complete information about *run-options* and *command-line-options* see the CONSOLE section in Command Line Reference.

Starting the Client Process

The LightWave Client process is a *server* application that runs on NonStop systems. It runs as either a Pathway serverclass or a standalone Guardian server process. The advantage of the Pathway option is Pathway's server process management and load-balancing features. In either case, LightWave Client supports a number of startup *command-line-options* that control various aspects of its operation. The same options are supported in both Pathway and standalone configurations, specified as server PARAMs or command-line-options, respectively. The object file name for LightWave Client is "CLIENT" and is located in the installation subvolume.

Starting LightWave Client as a Standalone Server Process

LightWave Client may be started by running the CLIENT program from TACL.

```
> run client / run-options / command-line-options
```

Examples

The example starts the CLIENT process as $LWC in CPU 0, supporting the service "MyService" located in filesystem $DATA.LWFS.

```
> run client / name $lwc, cpu 0 / --api MyService --base-url http://api.example.com
--log $zhome info --standalone
```

More Information

For complete information about *run-options* and *command-line-options* see the CLIENT section in Command Line Reference.

Starting LightWave Client as a Pathway Server

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

The LightWave Client Process may be configured as a Pathway server, which allows you to take advantage of Pathway's process management and load-balancing features. The configuration options are the same as those for standalone process configuration, however in the Pathway environment individual options are supplied as server PARAMs. The following is a sample pathway server configuration. Note that the server settings here (createdelay, deletedelay, maxservers, etc.) are merely examples, not requirements or recommendations.

```
reset server
set server cpus 0:1
set server createdelay 0 secs
set server deletedelay 60 secs
set server highpin on
set server linkdepth 1
set server maxservers 6
set server maxlinks 20
set server numstatic 0
set server program CLIENT
set server tmf on
set server debug off
set server param api "MyService"
set server param base-url "http://api.example.com"
set server param log "$0 info text"
add server MY-SERVICE
```

You may also provide the options in a command file by setting the SERVER STARTUP value to @<command-file>. The following is a sample pathway configuration using this technique.

```
reset server
set server cpus 0:1
set server createdelay 0 secs
set server deletedelay 60 secs
set server highpin on
set server linkdepth 1
set server maxservers 6
set server maxlinks 20
set server numstatic 0
set server program CLIENT
set server tmf on
set server debug off
set server startup @config
add server MY-SERVICE
```

Startup options supplied as PARAMs will override any options supplied with the STARTUP attribute or in a command file.

Stopping LightWave Client

A LightWave Client CONSOLE process may be shut down with the CONSOLE --shutdown command using the following syntax:

```
> run console --shutdown <console-process-name> [!]
```

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

A LightWave Client standalone CLIENT process should be stopped using the standard TACL STOP command. CLIENT processes configured as Pathway serverclasses should be stopped using Pathway serverclass shutdown procedures.

The LightWave Client Console

The LightWave Client 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 Client instance. Note that you must connect to the port that has been configured using the --console-ports option. For example, if your LightWave Client Console was started using the following command:

```
> run console / name $lw, 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
- APIs
- Diagnostic Logs
- Server Certificates
- HTTP Logs
- Users and Groups
- Managing the Filesystem

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 Client 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 Client instance. The Dashboard contains three sections.

Processes

This section shows the current status of the LightWave Client CONSOLE process. 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 CONSOLE 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.

APIs

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

Diagnostic Logs

Diagnostic Logs show detailed information about API requests to your LightWave Client instance. The logs are written to the Enscribe file system and may be viewed from the NonStop command line as edit files or viewed in the LightWave Client Console. 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.

> ⚠ Diagnostic logging is resource intensive and will degrade the performance of all services on your LightWave Client instance. Diagnostic logging should not be enabled on production instances unless necessary.

To view logs:

1. Enter the name of the subvol that contains the Diagnostic Logs.
2. The available logs are listed.
3. Select the log to view.

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
Serialize	The time between the receipt of the request IPM from the requester application and the completion of serialization of the IPM into the web service request path components, headers, and payload.
Connect	The time between the completion of serialization and the completion of the connection. This includes completing the socket connect and if applicable, completing the SSL handshake. To improve performance, CLIENT uses the HTTP Persistent Connections protocol to reuse connections for multple requests. If a connection was reused, the Connect time will be 0.

Time	Description
Request	The time between completion of the connection 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 has received the entire request. Delays in the network, including one or more proxies, may increase the time required for the web service to receive the entire request.
Response	The time between completion of the send and the network receive of the entire response. This includes web service processing time and the time required to transfer the response over the network.
Deserialze	The time between completion of the network receive and deserialization of the web service response headers and payload into the response IPM that is returned to the requester application.
Total	The total time between receiving the request IPM and returning the response IPM to the requester application.

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.

You may also import an existing server certificate and private key created from another source. The import file must be in PKCS12 format.

A trusted HTTPS connection requires a Server Certificate signed by a recognized Certificate Authority. For testing purposes, LightWave Client 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 services.

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 Client 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 Client 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 Client console ports. The ports must be configured at CONSOLE startup. Use the Common Name (see above) when specifying the port. See CONSOLE Command Line Options for more information about configuring TCP/IP ports.

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

HTTP Logs

HTTP Logs show HTTP request traffic to your LightWave Client Console. 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. 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.

Users and Groups

The User and Group management features allows you to create and manage LightWave Client users and groups.

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 is disabled.
7. 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 Client file system is the repository for all LightWave Client configuration data: it contains API definitions, Server Certificates, configured users and groups, and HTTP logs. The file system consists of four audited Enscribe files: FC00, FC00A, FC01 and FC02. A file system cannot be shared by more than one LightWave Client Console process. Refer to LWCCOM for more information about managing the file system.

Copying or Moving the File System

You can copy the LightWave Client filesystem by copying all of the FC0* 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 the files are not open.

```
> fup listopens $DATA.MYFS.FC0*
```

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

```
> pak myfspak, $DATA.MYFS.FC0*, audited, listall
> 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*.

```
> volume $DATA.MYNEWFS
> fup
- dup $DATA.MYFS.FC0*, *, saveall
- alter FC00, altfile (0, FC00A), audit
- alter FC00A, audit
- alter FC01, audit
- alter FC02, audit
- exit
```

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

Important

The filesystem contains server certificates (if installed) which contain information specific to the host name used to access LightWave Client Console. If copying the filesystem to another NonStop system, you may need to obtain and install a new server certificate for the new 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 FC00 & FC01 files over time by a housekeeping process within LightWave Client Console. You can use LWCCOM to cause the FC00 & FC01 data records to be immediately deleted by running the following command from the TACL prompt:

```
> run lwccom control filesystem <filesystem-subvol>, clean
```

After cleaning the filesystem there may still be a large amount of unused disk space allocated to the FC01 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:

```
> 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 the LightWave Client Console 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
--diag-log	Diagnostic dumping 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
- Diagnostic Log Configuration

Configuration File Format

Configuration files are Enscribe EDIT files and uses "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

# Configure diagnostic logging.
[diag-log]
enabled=1
subvol=$vol.subvol
```

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 CLIENT --log +$vol.subvol.logcfg [--monitor log:15 ] ...
> run CONSOLE --log +$vol.subvol.logcfg  [--monitor log:15 ] ...
```

Configuration Reference

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

Option	Descriptione
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	Descriptione
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 "trace" 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
```

Diagnostic Log Configuration

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

```
> run CLIENT --diag-log +$vol.subvol.logcfg  [--monitor diag-log:15 ] ...
```

Configuration Reference

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

Option	Description
enabled	The value may be "0" to disable message dumping or "1" to enable it. The default is "0".
subvol	Specifies the subvol in which the individual diagnostic log files will be created. If not specified the logs current subvol of the CLIENT program.
replyCodeFilter	A list of reply code values that should be selected for logging, specified as a range.
httpStatusFilter	A list of HTTP status values that should be selected for logging, specified as a range.
content	A list of selectors indicating the content that should be selected for logging. The following selectors are available: • default - Include all diagnostic information • basic - Include basic diagnostic information. Request and reply HTTP body and IPM data is omitted. • body - Include basic information and the request and reply HTTP body. • ipm - Include basic information and the request and reply IPM data. If omitted, "default" is used.

Examples

```
# Log to a file. Only log when the LightWave reply code is 1 or 2  or the HTTP status
code is 400-999
[diag-log]
enabled=1
subvol=$data1.diaglogs
replyCodeFilter=1,2
httpStatusFilter=400:999
content=body
```

Configuration Best Practices

Performance & Scalability

- Run the CLIENT process as a Pathway Server Class. See CLIENT Process Configuration.
- Use configuration files to specify logging options, which allows the options to be changed without re-configuring the server class. See Using Configuration Files.
- 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.
- When using TLS connections with HTTP Basic authentication, use the *pre-auth* option. See http-credentials.

Security

- Change the default administrator password!
- Use Server Certificates and configure only HTTPS console ports.
- Use TLS connections to services whenever possible, both when accessing REST applications and accessing the LightWave Client Console.
- Use the *sensitive* schema property to avoid disclosing sensitive data in logs. See Sensitive Data Masking.
- Use credentials files to supply configuration credentials. See Using Credentials Files.
- Use Guardian security to appropriately secure API, configuration, credential, and program files.
- Only install the Console in production environments when necessary, or only run the Console in production when necessary. The Console is a development tool and generally not necessary outside of the development environment.

Command Line Reference

The Command Line Reference describes all startup options for the programs supplied with the LightWave Client release.

Refer to the following sections for more information:

- CLIENT
- CONSOLE
- CUTILITY
- LWCCOM

CLIENT

CLIENT is the LightWave Client process, which is responsible for relaying messages between a client application and a web service based on a previously defined API Definition.

The process is started by running the CLIENT program from TACL or by configuring the program as a Pathway Server Class. This section describes how to start the program and available program options.

- Starting CLIENT as Standalone Process
- Configuring CLIENT as a Pathway Server Class
  - Configuration Using the Server Class STARTUP Attribute
    - Server Class configuration
    - Server Class configuration
  - Configuration using Server Class PARAMs
    - Server Class configuration
    - Server Class configuration
- CLIENT Program Options
- Remarks
- Examples
  - Standalone Process
    - Starting from TACL using options on the command line
    - Starting from TACL using a command file
    - Specify values for API parameters username and api-key

Starting CLIENT as Standalone Process

The CLIENT process may be started by running the CLIENT program from TACL.

```
> run CLIENT / run-options / program-options
```

The "--api" and "--base-url" command-line-options are required. All others are optional. The run-options are the standard TACL run options. Note that the process does not open the IN or OUT file. You should be logged-on as a user with sufficient privileges to access the system resources that the process requires.

Configuring CLIENT as a Pathway Server Class

The CLIENT process may be configured as a Pathway Server Class. This is the preferred method, as it allows CLIENT processes to be created and deleted to meet application demand. When configuring a Server Class, program options may be specified with the STARTUP attribute or specified as PARAMs.

Configuration Using the Server Class STARTUP Attribute

Program options may be supplied directly in the STARTUP string or entered into an EDIT file and supplied using a Command File.

Server Class configuration

```
reset server
set server program        client
set server startup        "--api myapi --base-url http://api.example.com"
```

Or

Server Class configuration

```
reset server
set server program        client
```

```
set server startup        "@cmdfile"

EDIT file cmdfile contents:
--api myapi
--base-url https://api.example.com
--log logfile info
```

Using the STARTUP attribute with a command file allows the program options to be modified without re-configuring the Server Class. Note that changes to the command file take effect when a new CLIENT process is started and have no effect on processes that are already running.

Configuration using Server Class PARAMs

Program options may be specified as individual PARAMs. When options are specified as PARAMs, do not include the leading '--' characters in the PARAM name which would make the name invalid. The PARAM values should be enclosed in quotes:

Server Class configuration

```
set server param api        "myapi"
set server param base-url   "http://api.example.com"
```

Some program options, such as *cert-no-verify*, do not require a value. The presence of these options on the command line activates the feature. Because Pathway PARAMs require a value, when specifying these option as a PARAM, specify any value for the PARAM to activate the feature. The value itself is ignored. For example:

Server Class configuration

```
set server param cert-no-verify "1"
set server param cert-no-verify "true"
```

Because the PARAM value is ignored, both of these examples will activate the *cert-no-verify* feature.

Startup options supplied as PARAMs will override any options supplied with the STARTUP attribute or in a command file.

CLIENT Program 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).

--api <file-name>

The name of an API definition file. This option is required. The API definition must be exported from the LightWave Client filesystem into an API definition file through the LightWave Client Console or by using the CUTILITY -- export-api command. Changes to API definition can be incorporated while CLIENT is running (i.e., without restarting) if the `--monitor api` option is used.

--api-param-<param-name> <param-value>

The API parameter value for the parameter <param-name>. This option should be specified for each parameter defined in the API definition.

--auth <file-name>

The name of an auth config file. For more information on auth configuration and request signing, see Request Authentication and Signing.

--base-url <url>

The base URL of the target web service in the form *http[s]://host[:port]/[base-path]*. Note that the optional base-path will be concatenated with the API operation path to form the full URL. This option is required.

--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 CLIENT 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 CLIENT process. Note that client applications are responsible for disposing of the BLOB files once they have been processed.

--ca-root-certs <file-name>

The name of the file containing the certificates of all trusted root certificate authorities. If omitted, this value defaults the CAROOT file that is provided with the LightWave Client software.

--ca-local-certs <file-name>

The name of a file containing the certificates of local trusted certificate authorities. If omitted, no local CA certificates are loaded.

--cert-no-verify

The presence of this option indicates that the CLIENT process should not validate the server certificate for common name, expiration date, or issuer when an secure connection is established. Note that this option should only be used in a development/ test environment where the server may not necessarily pass all of the verification criteria. It should not be used in a production environment.

--client-cert <certificate-file> [{ <pass phrase> | +<credentials-file> }]

The client certificate to use for the secure connection. Specify the name of the file and, if required, the associated pass phrase required to access the certificate. Specify the pass phrase, in plain text, or the name of an existing LightWave Credentials file containing the encrypted pass phrase.

--default-encoding <encoding-name>

Specified the default encoding to use for character string conversions for the API. 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.

--diag-log <config-spec> | +<diag-log-config-file> | <subvolume-name>

Enables diagnostic logging and specifies the subvolume where the logs are stored, the location of a log config file, or a string consisting of diagnostic log configuration options. Log files are named using the format DL*nnnnnn* where nnnnnn is a sequence number. The logs may be viewed using command line tools or from the LightWave Client Console if it is installed. See Diagnostic Log Configuration for information about diagnostic logging configuration files and config-spec options. Note that the <subvolume-name> option is available for compatibility with release prior to 1.0.5. Using a config-file or config-spec is recommended.

--http-credentials { <userid>:<password> | +<credentials-file> } [pre-auth]

The credentials required for the HTTP connection to the Web service host when HTTP Basic or Digest authentication is required. Specify either your userid and password (in plain text) or the name of an existing credentials file containing the encrypted userid and password. See Using Credentials Files for information about creating credentials files. Supplying "pre-auth" indicates that pre-authentication should be used. This causes Basic authentication credentials to be sent with every request, without waiting for an HTTP 401 response. Use of this option can improve performance on connections that use HTTP Basic authentication but has serious security implications

for non-HTTPS (TLS) connections. The "pre-auth" option should not be used unless the security implications are fully understood.

--http-proxy-host <address[:port]>

The host name or IP address and port of an HTTP proxy that should be used for HTTP/HTTPS connections. If omitted, no proxy is used. If the port value is omitted, port 80 is used.

--http-proxy-credentials { <userid>:<password> | +<credentials-file> }

The credentials required for the HTTP proxy. Specify either your userid and password (in plain text) or the name of an existing credentials file containing the encrypted userid and password. See Using Credentials Files for information about creating credentials files.

--http-request-timeout <milliseconds> [!]

The number of milliseconds to wait for a web service request/response exchange to complete. If omitted, the default value of 60 seconds (60000 milliseconds) is used. If '!' is specified, this value will override any value set by the client application in the rq_timeout field of the LightWave request header. If '!' is not specified, and the client application specifies a timeout value in the rq_timeout field, the value in the rq_timeout field is used.

--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.

--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: api, log, diag-log. See Using Configuration Files for information about monitoring log and diag-log configuration files.

--standalone

The presence of this option causes the process to ignore close messages. In order to function properly in the Pathway environment, the CLIENT process, by default, matches open and close messages and terminates when all clients have closed the process. This option can be used to prevent the process from terminating when it is run as a standalone process.

--string-padding

Specifies a string padding value that will override the string padding setting in the API definition. The value may be 'zeros', 'spaces', or an integer in the range 0 - 255.

--tcpip-bind-addr <ipv4-address>

Specifies an IPv4 address to bind to for TCP/IP connections. This option may be used to specify the IP address from which connections will originate, when a TCPIP provider is configured with multiple IP addresses. If omitted, the default IP address for the TCPIP provider is used.

--tcpip-process <process name>

Specifies the name of the TCPIP process that the process should use. If omitted, the value of the =TCPIP^PROCESS^NAME MAP define is used if it exists, otherwise $ZTC0 is used.

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

Specifies the list of protocols that should be available for TLS connections. If omittied, 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	4

Z-VERSION	Product major version

Examples

Standalone Process

Starting from TACL using options on the command line

```
> run CLIENT / name $lwc, nowait, term $zhome, cpu 0 / --standalone --api employee --
base-url http://api.example.com --log $0 info
```

Starting from TACL using a command file

```
> run CLIENT / name $lwc, nowait, term $zhome / @cmdfile

EDIT file cmdfile contents:
--standalone
--api employee
--base-url http://api.example.com
--log $0 info
```

Specify values for API parameters username and api-key

```
> run CLIENT / name $lwc, nowait, term $zhome / --standalone --api employee --base-
url http://api.example.com --api-param-username johnsmith --api-param-api-key
35bddf603d7b4cef9fbaf1689c1cd49e
```

Monitor the API and log configuration files for changes

```
> run CLIENT / name $lwc, nowait, term $zhome / --standalone --api employee --base-
url http://api.example.com --log +logcfg --monitor api:30 log:15
```

Pathway Server Class Using STARTUP Attribute

Server Class configuration

```
== Settings for linkdepth, maxservers, maxlinks, etc are examples, not
recommendations.
reset server
set server cpus           0:1
set server createdelay    1 secs
set server deletedelay    120 secs
set server highpin        on
set server linkdepth      10
set server maxservers     10
set server maxlinks       10
set server numstatic      0
set server program        client
set server tmf            off
set server debug          off
set server startup        "@cmdfile"
add server example

EDIT file cmdfile contents:

--api employee
--base-url http://api.example.com
--log +logcfg
--monitor api:30 log:15
```

Pathway Server Class Using PARAMs

Server Class configuration

```
== Settings for linkdepth, maxservers, maxlinks, etc are examples, not
recommendations.
reset server
set server cpus           0:1
set server createdelay    0 secs
set server deletedelay    15 secs
set server highpin        on
set server linkdepth      5
set server maxservers     1
set server maxlinks       20
set server numstatic      0
set server program        client
```

```
set server tmf            off
set server debug          off
set server param          api "employee"
set server param          base-url "https://api.example.com"
set server param          diag-log "+logconf"
set server param          log "+logconf"
set server param          monitor "api:5 diag-log:5 log:5"
set server param          tcpip-process "$ztc0"
add server example
```

CONSOLE

CONSOLE is the LightWave Client Console process, which supports the administrative and management user interface for LightWave Client.

Starting the Process

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

```
> run CONSOLE / run-options / command-line-options
```

The "--filesystem" and "--console-ports" command-line-options are required. All others are optional.

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).

--backupcpu <cpu-number>

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

--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. This option is required. See Certificate Management for information about installing server certificates.

--filesystem <subvolume-spec>

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

--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.

--sts-max-age <seconds>

Enables the Strict-Transport-Security header 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 omittied, the default value is "TLSv1.2 TLSv1.3".

Examples

Start the Console 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. This example assumes that a server certificate with Common Name `lightwave.example.com` has already been installed.

```
> run console / name $LWCON, cpu 0 / --backupcpu 1 --console-ports $ztc0:8080
$ztc0:8443:lightwave.example.com
```

CUTILITY

CUTILITY has been deprecated and will no longer be updated. LWCCOM has replaced CUTILITY as the LightWave Client management CLI.

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

```
> run CUTILITY / run-options / command-line-options
```

Important

Always run CUTILITY with the --update-filesystem option after installing a new version of LightWave Client.

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).

--api-to-ddl <api-file-name> <ddl-file-name> [!]

Generates a DDL source file from an API definition stored in an Enscribe file. Specify the '!' option to indicate that the output DDL source file should be replaced if it exists. Note that this feature became available in release 1.0.1.

--check-cf <file-name>

Validates the specified credentials file.

--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 CONSOLE. 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-cf <output-credentials-file> [!] <credentials-string> | +<credentials-input-file> | '?' [<target-file-name>]

Creates an encrypted LightWave Client credentials file. See Using Credentials Files for more information about credentials files. The <output-credentials-file> parameter is the name of the credentials file to create. Use the '!' character to specify that the file should be overwritten if it already exists. The input credentials may be supplied on the

command line, supplied in an edit file by prefixing the name of the file with '+', or you may be prompted for the credentials by specifying '?'. When prompted, the credentials are not displayed on the terminal.

As an added security measure, the credentials file contains the name and location of the file itself. Thus, a credentials file will be invalid if renamed or moved to a new location. To create a credentials file for use in another location (e.g. another system), use <target-file-name>. When specified, the credentials file will be created such that it will be valid when copied to the target file name.

--create-filesystem

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

--export-api <api-name> <output-api-file> [!]

Exports an API definition from the filesystem to an Enscribe file. Specify the '!' option to indicate that the output Enscribe file should be replaced if it exists. Note that when using this feature, if the --filesystem option is not specified, the filesystem location is assumed to be the current subvol. Note that this feature became available in release 1.0.1.

--filesystem <subvolume-spec>

Specifies the subvolume location of the target LightWave Client 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.

--import-api <input-api-file> < <api-name> | '*' > [!]

Imports an API definition stored in an Enscribe file into the filesystem. If '*' is specified as the API name, the API stored in the definition will be used. Specify the '!' option to indicate that an existing API with the same name should be replaced if it exists. Note that when using this feature, if the --filesystem option is not specified, the filesystem location is assumed to be the current subvol. Note that this feature became available in release 1.0.1.

--set-password <username> <password>

Overwrites the password for the named user.

--update-filesystem

Updates a LightWave Client filesystem in the location specified by the --filesystem option. The update process uses the file named CCPnnnnn (where nnnnn is a version number) in the installation subvolume to update the Console web application. If there is more than one CCPnnnnn file, the latest version is used. The update process may make the filesystem incompatible with earlier releases of LightWave Client.

LWCCOM

The LWCCOM command line interface (CLI) can be used as an alternative to the LightWave Client Console for management of a LightWave Client instance. Commands may be entered interactivity or supplied with a TACL macro or obey file.

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

```
> lwccom
LightWave Client COM
Copyright (c) 2024 NuWave Technologies, Inc. All rights reserved.
LWCCOM 1-> help
Help is available on the following commands and objects:
Commands:

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

Objects:

API                 CERTIFICATE         CONFIG              CREDENTIALS
FILESYSTEM          GROUP               LICENSE             USER

Enter HELP <command> | <object> for more information.
LWCCOM 2-> help api
The following commands can be used with the API object:

ALTER               DELETE              EXPORT              IMPORT
INFO

Enter HELP <command> API for more information.
LWCCOM 3-> help alter api

ALTER API Command

Alters an API in the filesystem

  ALTER API <api-name>
        [ , NAME <api-name> ]
        [ , DESCRIPTION <string> ]

<api-name>
```

```
  The name of the API.

NAME <api-name>

  New name for the API.

DESCRIPTION <string>

  New description for the API.

LWCCOM 4->
```

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
CLIENT	NUWAVE	4	1	CLIENT
CONSOLE	NUWAVE	4	1	CONSOLE

CLIENT 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 not occur and should be reported to the LightWave administrator.

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.

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.

1211

```
ERROR Required startup option --api was not supplied.
```

Cause

The required startup option was not supplied.

Effect

The process cannot start.

Recovery

Restart the process with all required options.

1212

```
ERROR Required option --base-url was not supplied.
```

Cause

The required startup option was not supplied.

Effect

The process cannot start.

Recovery

Restart the process with all required options.

1213

```
ERROR The --api option value value is invalid. The value is not a valid file name.
```

Cause

The specified option value is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1214

```
ERROR The API file file-name cannot be opened: reason.
```

Cause

The API file could not be opened for the reason indicated.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1215

`ERROR The API file file-name does not contain valid JSON. Parser error at line line, column column, reason.`

Cause

The file does not contain valid JSON and cannot be a valid API definition.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1216

`ERROR API file file-name does not contain a valid API definition.`

Cause

The file does not contain a valid API definition.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1217

`ERROR The --base-url 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.

1218

`ERROR The --http-credentials option value is invalid. Unrecognized option value.`

Cause

The specified option value is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1219

`ERROR The --ca-root-certs option value value is invalid. The value is not a valid file name.`

Cause

The specified option value is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1220

`ERROR The --ca-root-certs option value value is invalid. The file file-name does not exist or access error.`

Cause

The specified option value is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1221

`ERROR The --ca-local-certs option value value is invalid. The value is not a valid file name.`

Cause

The specified option value is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1222

`ERROR The --ca-local-certs option value value is invalid. The file file-name does not exist or access error.`

Cause

The specified option value is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1223

`ERROR The --client-cert option value value is invalid. The value is not a valid file name.`

Cause

The specified option value is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1224

`ERROR The --client-cert option value value is invalid. The file file-name does not exist or access error.`

Cause

The specified option value is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1225

`ERROR Unable to load the client certificate from file file-name, the certificate is not valid or the passphrase is incorrect.`

Cause

The specified certificate file is not valid or the supplied passphrase is incorrect.

Effect

The process cannot start.

Recovery

Correct the file name and passphrase and restart the process.

1226

`ERROR The --monitor 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.

1229

```
ERROR The --diag-log 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.

1230

```
ERROR Diagnostic log config file file-name cannot be loaded.
reason
```

Cause

The diagnostic log config file is being monitored, but it could not be loaded for the reason shown.

Effect

The new configuration was not loaded. The existing configuration remains in effect.

Recovery

Correct the configuration and wait until the reload occurs.

1232

```
ERROR tcpip-process-configuration-error
```

Cause

The TCP/IP process configuration is invalid for the reason shown.

Effect

The process cannot start.

Recovery

Correct the --tcpip option value and restart the process.

1234

```
ERROR The --http-request-timeout option value value is invalid.
The timeout value must be in the range range milliseconds.
```

Cause

The specified option value is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1237

```
ERROR The --blob-files 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.

1238

```
ERROR Error file-system-error occurred while attempting to create
diagnostic log file file-name.
```

Cause

The indicated file system error occurred while creating a diagnostic log file. The error may be due to a volume full condition or a security violation.

Effect

The diagnostic log was not created.

Recovery

Correct the underlying condition.

1239

```
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.

1240

```
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.

1241

```
ERROR The --string-padding option value value is invalid. The value must be 'spaces', 'zeros', or an integer in the range 0 - 255.
```

Cause

The specified option value is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1242

```
ERROR The --http-proxy-host option value value is invalid.
```

Cause

The specified option value is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1245

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

Cause

The specified option value is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1246

```
ERROR The --tls-protocols 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.

1254

`ERROR The --http-proxy-credentials 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.

1255

`ERROR The --http-credentials 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.

1257

`ERROR The TCPIP bind address ip-address is invalid. reason`

Cause

The specified option value is invalid.

Effect

The process cannot start.

Recovery

Correct the option value and restart the process.

1258

```
ERROR auth-config-error
```

Cause

The indicated error occurred while loading the authentication configuration file.

Effect

The process cannot start.

Recovery

Correct the configuration file 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 name 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 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.

1274

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

Cause

Excessing 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.

2208

```
WARN icu-load-error. Character encoding is limited to ISO-8859-1.
```

Cause

The ICUDATA file could not be loaded due to the indicated reason.

Effect

Character encoding is limited to ISO-8859-1.

Recovery

If other characer encodings are necessary, correct the underlying condition and restart the process.

2209

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

Cause

This event is used for informational purposes.

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.

3204

`INFO Stopped.`

Cause

The process has stopped.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3206

`INFO Startup options: startup-options`

Cause

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.

3209

`INFO Loaded API definition from file-name`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3210

`INFO API is api-name, api-timestamp.`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3211

`INFO Using Base URL base-url`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3212

`INFO Loaded HTTP credentials. Pre-authentication is [not] enabled.`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3213

`INFO Loaded count root CA certificates from file-name.`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3214

`INFO Loaded count local CA certificates from file-name.`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3215

```
INFO Loaded client certificate from file-name.
```

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3216

```
INFO Configuration monitoring is disabled.
```

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3217

```
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.

3219

```
INFO Loaded diagnostic log config from file-name.
```

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3220

```
INFO Diagnostic log config is diagnostic-log-config
```

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3221

```
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.

3222

```
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.

3223

`INFO Using TCPIP process process-name`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3224

`INFO Server certificates will [not] be verified.`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3225

`INFO HTTP request timeout is value milliseconds. Application override is enabled | disabled.`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3226

`INFO Loaded count API params: param-list`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3227

`INFO BLOB file config is 'pattern=pattern user=user security=security-string extents=n,n,n`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3228

`INFO Using TLS options: options`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3229

`INFO TLS cipher list: cipher-list`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3231

`INFO Loaded auth config from file-name`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3232

`INFO API string padding set to value`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3233

`INFO Using HTTP proxy host host-name[:port]`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3234

`INFO Loaded HTTP proxy credentials.`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3235

`INFO Using default encoding encoding`

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3238

`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.

3239

```
INFO Using TCPIP bind address ip-address
```

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.

3241

```
INFO Started.
```

Cause

The process has started successfully.

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.

CONSOLE 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 n be reported to the LightWave Client system administrator.

1110

`ERROR port-type port tcpip-process:port bind failed, error 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, error 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, error 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-for-failure.
```

Cause

The filesystem in the specified *subvol-name* count 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 subvol is already opened by CONSOLE process process-name. CONSOLE processes cannot share a filesystem.`

Cause

A CONSOLE process was started using a filesystem in *subvol* than is already in use by another CONSOLE process. Filesystems cannot be shared among multiple CONSOLE processes.

Effect

The process stops.

Recovery

Start the CONSOLE 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.

1226

```
ERROR The --monitor 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.

1239

```
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.

1240

```
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.

1246

```
ERROR The --tls-protocols 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.

1256

```
ERROR Index error for filesystem filesystem-subvolume. 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

The process cannot start.

Recovery

Correct the index issue 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 name 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.

1273

```
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 as 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.

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.

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

2256

```
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.

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.

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.

3216

```
INFO Configuration monitoring is disabled.
```

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3217

```
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.

3221

```
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.

3222

```
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 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.

3229

```
INFO TLS cipher list: cipher-list
```

Cause

This event is used for informational purposes.

Effect

None.

Recovery

Informational message only; no corrective action is needed.

3238

```
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.

3240

```
INFO Shutdown started.
```

Cause

Process shutdown has started.

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 Client requires the following HPE 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.
- HPE NonStop TCP/IP
- HPE NonStop Transaction Management Facility (TMF), on systems running the CONSOLE process.

Although not required, LightWave Client integrates with these products, if installed:

- HPE NonStop TS/MP (Pathway). Large message support is available when supported by TS/MP.
- HPE Data Definition Language (DDL or DDL2).
- HPE 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

Upgrade Considerations

- When upgrading from a previous release the filesystem update procedure must be performed. See Updating LightWave Client for more information.
- The CUTILITY program has been deprecated. All CLI based product management should be done using LWCCOM.

Cumulative Change Log

1.0 Releases

1.0.7 - 27 Mar 2021

Problems Corrected

[LC-921] - --blob-files option is rejected if it contains more than 2 optional values
[LC-934] - CLIENT may abend when writing diagnostic log after request timeout
[LC-935] - CLIENT ignores socket connection reset while receiving response body on TLS connection
[LC-947] - Diag log system info RVU is incorrect
[LC-949] - DDL generator does not treat ENVIRONMENT as COBOL reserved word
[LC-951] - Fix for LC-917 broke request signing support
[LC-965] - LWCCOM pagesize command may report valid input as invalid
[LC-974] - Setting isSet: true does not work for JSON object serialization when the object is empty

1.0.6 - 28 AUG 2020

Problems Corrected

[LC-293] - API editor swagger import: vague mixed content error when Console is accessed via HTTPS and swagger is HTTP
[LC-844] - CLIENT may abend while parsing digest auth header with specific format
[LC-853] - HTTP log file name sequence does not roll over when date changes
[LC-854] - HTTP log version & date directives have incorrect values
[LC-855] - Console HTTP log time ranges should be in GMT to match the log timezone
[LC-874] - A structure element in an array of structures is not serialized if the element

is hidden in a previous structure in the array
[LC-875] - Diagnostic log is corrupted if line break occurs at multi-byte character
[LC-881] - HSTS header may be sent intermittently
[LC-888] - --log startup option ignores "event" format specification
[LC-892] - Scaled integers with values exceeding 12 bytes are serialized as "**********"
[LC-893] - unsigned long long values > 9223372036854775807 are not de/serialized correctly
[LC-900] - CONSOLE does not detect when filesystem is not TMF audited, resulting in unpredictable behavior
[LC-904] - Process logging configured at debug level may expose sensitive data in HTTP headers

New Features

[LC-880] - Implement XSRF protection in the Console

1.0.5 - 04 MAR 2020

Problems Corrected

[LC-497] - At sign '@' not parsed correctly when used as option value.
[LC-654] - LWCCOM import cert does not replace existing cert with same common name.
[LC-667] - CONSOLE does not shut down when filesystem in use detected.
[LC-673] - Certificate installation dialogs with textarea fields have no Cancel / Install buttons
[LC-678] - Chunked encoded response is not processed correctly after HTTP 401 challenge
[LC-711] - CLIENT may abend if response payload is larger than IPM blob size
[LC-712] - API editor path field requires leading slash and disallows trailing slash
[LC-714] - CLIENT may abend on startup if client certificate file cannot be opened
[LC-718] - Certificate import with duplicate CN does not replace existing certificate.
[LC-719] - Integer with scale values less than 0.1 are serialized as scientific notation
[LC-723] - Schema editor syntax check no longer works.
[LC-726] - Schema editor error tooltip is not always visible
[LC-770] - CLIENT may abend when configured with invalid base-url
[LC-771] - Log configuration file documentation shows invalid level option "trace"
[LC-772] - INVALID-TYPE error is not documented
[LC-777] - Vague error "Deserialization error: Unexpected json type 0" is reported when the payload contains an object when a primitive is expected.
[LC-778] - Azure IoT Hub request signature algorithm leaks memory.
[LC-779] - String encoding validation leaks memory.
[LC-782] - Deserialization error occurs when the wildcard response is not the last response defined for the operation
[LC-783] - INVALID-HDR-VERSION error (103) is missing from documentation and generated DDL

[LC-784] - Credentials file error is logged with incorrect error code.
[LC-785] - API editor does not refresh when a new version of an API is uploaded
[LC-791] - CLIENT does not shut down when API cannot be opened.
[LC-792] - Intermittent ICU load errors on TNS/E
[LC-801] - CLIENT process leaks memory on each request
[LC-802] - Organization field missing in generated CSR
[LC-808] - Client application receives connection reset or timeout error when HTTP 401 response is received and connection is closed
[LC-814] - Console uses mixed content links in page footer
[LC-815] - Upgrade AngularJS to version 1.7.9 to mitigate security vulnerabilities
[LC-837] - Response length is incorrect when response is not chunked encoded and no Content-Length header is returned
[LC-840] - HTTP authentication pre-auth does not work on non-secure (HTTP) connection

New Features

[LC-794] - Upgrade to OpenSSL 1.0.2.t
[LC-816] - Allow selection of diagnostic log content using diagnostic log configuration. See Diagnostic Log Configuration
[LC-824] - Add isSet schema property to indicate presence of optional elements. See Working with Optional Elements
[LC-833] - Add hideIfEmpty schema property to omit serialization of empty elements. See Working with Optional Elements

1.0.4 - 05 MAR 2019

Problems Corrected

[LC-463] - API import accepts any valid JSON file
[LC-468] - Application-supplied Authorization header overwritten when --http-credentials specified
[LC-527] - CLIENT returns parser error for BLOB response.
[LC-532] - CUTILITY -cf report internal errors 9 when there is no whitespace between file name and dammit (!)
[LC-535] - API editor delete method no longer works in Firefox
[LC-542] - Slashes appended to base-url result in multiple slashes on the URI.
[LC-547] - AWS signature is invalid when header, query param, or URI contain embedded spaces.
[LC-554] - Console may log out due to inactivity while using the API editor for a long period without saving.
[LC-558] - ISO8601:full-date support is not available in 1.0.x CLIENT
[LC-565] - LWCCOM create credentials only works when user is prompted for credentials
[LC-579] - DDL generator creates invalid DDL name when the original name contains digits and is longer than 26 characters

[LC-580] - Authorization header is not masked in the diagnostic log
[LC-588] - Swagger import may generate incorrect lengths for path elements/params/headers
[LC-606] - Diagnostic log sensitive data masking warning is not consistently displayed
[LC-608] - Reply buffer reserved area is not null filled.
[LC-609] - Command line error reported when argument contains double dash
[LC-610] - API editor path dialog hangs browser when certain URIs with path components are entered
[LC-613] - Command line consisting only of a comma is reported as command abbreviation
[LC-614] - Diagnostic log delete buttons are enabled when there are no logs
[LC-612] - Diagnostic log is not written when the request is cancelled

New Features

[LC-615] - Console returns X-Content-Type-Options, X-Frame-Options, X-XSS-Protection, Strict-Transport-Security security headers.
[LC-628] - Add options to disable TLS 1.0 & 1.1

1.0.3 - 05 OCT 2018

Problems Corrected

[LC-509] - Proxy-Authorization header is sent unnecessarily when http-credentials with pre-auth option is used
[LC-513] - Measure counters are inaccurate when CLIENT is processing multiple concurrent requests
[LC-517] - Request cancellation may cause other outstanding requests to fail with socket errors.

New Features

[LC-411] - Add support for international character set encoding.
[LC-450] - Add support for sensitive data masking.
[LC-493] - Allow specification of extents for BLOB files
[LC-511] - Updated CAROOT
[LW-748] - Add LWCCOM CLI utility

1.0.2 - 11 APR 2018

Problems Corrected

[LC-280] - CUTILITY abend on --set-password when userid / password not provided
[LC-382] - CLIENT generates invalid AWS signature when AccessKeyId not supplied
[LC-386] - CLIENT returns JSON parser error when content-type header indicates JSON content but the response body is empty
[LC-387] - CLIENT serializes type "longlong" as a string
[LC-389] - Cannot view log files via Console if log files moved from original location

[LC-391] - CLIENT abends when field overflow occurs on reply header.
[LC-395] - CUTILITY --set-password gives "internal error" when supplied username does not exist
[LC-396] - CUTILITY --set-password accepts; ignores certain characters in passwords
[LC-399] - CLIENT reports SSL shutdown that occurs during request as a request timeout
[LC-443] - Debug log shows SSL errors when used with IE 11.
[LC-446] - dependsOn field that is an element of an array item is only set on the first element of the array
[LC-447] - Console times out unexpectedly
[LC-470] - CLIENT shuts down connection when http proxy responds to tunnel request with Connection: close header.
[LC-471] - CLIENT option --cert-no-verify is not working

New Features

[LC-473] - Add custom measure counters.
[LC-484] - Upgrade to OpenSSL 1.0.2o

1.0.1 - 14 NOV 2017

Problems Corrected

[LC-286] - Response header list can be truncated if the headers arrive in more than one TCP/IP recv buffer
[LC-306] - stringPadding set at the API level is not honored
[LC-308] - DDL generator emits incorrect PIC for numeric sign leading/trailing fields
[LC-309] - API Editor: Add schema dialog close throws js exception
[LC-311] - DDL generator may emit DDL with incorrect dependency order
[LC-314] - DDL generator emits incorrect DDL for timestamp type.
[LC-315] - DDL generator may emit DDL with duplicate object names
[LC-319] - API export creates an invalid output edit file if lines in the API definition exceed 239 characters.
[LC-322] - Host names that begin with a digit are not resolved correctly.
[LC-323] - HTTPS connections hang when a connection failure occurs
[LC-324] - Float/double mapping not working
[LC-331] - API Editor: schema editor does not allow types "unsignedLonglong", "numericSignTrailing", et al
[LC-335] - The "size" property for integer types declared in schema is ignored. The default size for the type is used.
[LC-337] - Last character of reply error message is truncated on response not defined error (code 12).
[LC-338] - API import from Swagger does not handle responses typed with $ref
[LC-340] - Strings are transferred to/from the message payload in UTF-8
[LC-341] - API Editor: allows response mapping with missing source
[LC-343] - Request timeout occurs when service returns HTTP status 204 (no content).

[LC-344] - CLIENT process shutdown is slow when socket connections are cached.
[LC-350] - Export DDL crashes when API path or schema is 'too long'
[LC-354] - JSON by example converts a value of 'null' to 'boolean', s/b "string"
[LC-356] - Diagnostic log deserialization time is invalid when a deserialization error occurs.
[LC-377] - sizeIs not working for types base64 and hexBinary
[LC-316] - API Editor: correct schema generation for date/time elements
[LC-359] - Diagnostic log config is logged as empty when all diag-log config options are commented

New Features

[LC-276] - Implement HTTP proxy support
[LC-317] - Add support for ISO8601 full-date
[LC-339] - Add support for cloud service signatures (AWS, Azure)
[LC-228] - Add API and DDL export functions to CUTILITY
[LC-271] - Add a sample startup file for server and console
[LC-330] - Allow API string padding to be set using CLIENT startup option
[LC-342] - API Editor: Allow path spec to be only forward slash.
[LC-351] - Add API filename to diagnostic log output
[LC-355] - API Editor: expand initial response to show data mapping; do not add (incomplete) mapping entry by default

1.0.0 - 15 JUN 2017

This is the first GA release of LightWave Client.

1.1 Releases

1.1.3 - 27 Mar 2021

Problems Corrected

[LC-917] - CLIENT may send an invalid Authorization header when a cached connection is reset by the web service provider
[LC-921] - --blob-files option is rejected if it contains more than 2 optional values
[LC-934] - CLIENT may abend when writing diagnostic log after request timeout
[LC-935] - CLIENT ignores socket connection reset while receiving response body on TLS connection
[LC-942] - API Editor allows download and DDL generation when API schema has changed but has not been saved
[LC-949] - DDL generator does not treat ENVIRONMENT as COBOL reserved word
[LC-965] - LWCCOM pagesize command may report valid input as invalid
[LC-969] - OpenAPI 3 import produces schema with invalid type references
[LC-974] - Setting isSet: true does not work for JSON object serialization when the

object is empty
[LC-982] - OpenAPI 3 import incorrectly imports arrays of objects.
[LC-983] - Schema add by XML example does not work in IE11
[LC-985] - Update user profile sends credentials in the clear

1.1.2 - 28 AUG 2020

Problems Corrected

[LC-293] - API editor swagger import: vague mixed content error when Console is accessed via HTTPS and swagger is HTTP
[LC-570] - Schema editor XML add by example feature may report error "ReferenceError: soureAttribute is not defined"
[LC-844] - CLIENT may abend while parsing digest auth header with specific format
[LC-853] - HTTP log file name sequence does not roll over when date changes
[LC-854] - HTTP log version & date directives have incorrect values
[LC-855] - Console HTTP log time ranges should be in GMT to match the log timezone
[LC-874] - A structure element in an array of structures is not serialized if the element is hidden in a previous structure in the array
[LC-875] - Diagnostic log is corrupted if line break occurs at multi-byte character
[LC-881] - HSTS header may be sent intermittently
[LC-888] - --log startup option ignores "event" format specification
[LC-892] - Scaled integers with values exceeding 12 bytes are serialized as "**********"
[LC-893] - unsigned long long values > 9223372036854775807 are not de/serialized correctly
[LC-900] - CONSOLE does not detect when filesystem is not TMF audited, resulting in unpredictable behavior
[LC-904] - Process logging configured at debug level may expose sensitive data in HTTP headers

New Features

[LC-880] - Implement XSRF protection in the Console
[LC-895] - Add option to allow binding to a specific IP address

1.1.1 - 04 MAR 2020

Problems Corrected

[LC-497] - At sign '@' not parsed correctly when used as option value.
[LC-654] - LWCCOM import cert does not replace existing cert with same common name.
[LC-718] - Certificate import with duplicate CN does not replace existing certificate.
[LC-763] - Sensitive data in XML payloads is not masked in diagnostic logs.
[LC-765] - JSON serializer abends if @XmlAttribute annotation is on the schema element
[LC-770] - CLIENT may abend when configured with invalid base-url

[LC-771] - Log configuration file documentation shows invalid level option "trace"
[LC-772] - INVALID-TYPE error is not documented
[LC-777] - Vague error "Deserialization error: Unexpected json type 0" is reported when the payload contains an object when a primitive is expected.
[LC-778] - Azure IoT Hub request signature algorithm leaks memory.
[LC-779] - String encoding validation leaks memory.
[LC-782] - Deserialization error occurs when the wildcard response is not the last response defined for the operation
[LC-783] - INVALID-HDR-VERSION error (103) is missing from documentation and generated DDL
[LC-784] - Credentials file error is logged with incorrect error code.
[LC-785] - API editor does not refresh when a new version of an API is uploaded
[LC-792] - Intermittent ICU load errors on TNS/E
[LC-801] - CLIENT process leaks memory on each request
[LC-802] - Organization field missing in generated CSR
[LC-808] - Client application receives connection reset or timeout error when HTTP 401 response is received and connection is closed
[LC-814] - Console uses mixed content links in page footer
[LC-815] - Upgrade AngularJS to version 1.7.9 to mitigate security vulnerabilities
[LC-837] - Response length is incorrect when response is not chunked encoded and no Content-Length header is returned
[LC-840] - HTTP authentication pre-auth does not work on non-secure (HTTP) connection

New Features

[LC-794] - Upgrade to OpenSSL 1.0.2.t
[LC-816] - Allow selection of diagnostic log content using diagnostic log configuration. See Diagnostic Log Configuration
[LC-824] - Add isSet schema property to indicate presence of optional elements. See Working with Optional Elements
[LC-833] - Add hideIfEmpty schema property to omit serialization of empty elements. See Working with Optional Elements

1.1.0 - 03 JUL 2019

Problems Corrected (since 1.0.4 release)

LC-667 - CONSOLE does not shut down when filesystem in use detected.
LC-673 - Certificate installation dialogs with textarea fields have no Cancel / Install buttons
LC-678 - Chunked encoded response is not processed correctly after HTTP 401 challenge
LC-711 - CLIENT may abend if response payload is larger than IPM blob size
LC-712 - API editor path field requires leading slash and disallows trailing slash
LC-714 - CLIENT may abend on startup if client certificate file cannot be opened
LC-719 - Integer with scale values less than 0.1 are serialized as scientific notation

LC-723 - Schema editor syntax check no longer works.
LC-726 - Schema editor error tooltip is not always visible

New Features

Support for XML payloads.
Performance and memory use improvements for file BLOBs.
Support for swagger import in YAML format.
Support for CyberSource HTTP signature authentication.

1.2 Releases

1.2.9 - 15 July 2025

Problems Corrected

- Console updated to allow import of Openapi 3.1 definitions.
- Certificate credentials file errors properly handled.
- Nested XML array deserialization no longer stops at first element.

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.

1.2.8 - 24 Sep 2024

Problems Corrected

- Schema editor does not recognize type attribute ddlIsElementaryType.
- DDL export may become unresponsive when schema type names are very long.
- Serialization error occurs when a query param or form field name includes '.'.
- ERROR event 1202 is logged at INFO level.
- INFO event 3232 is logged at ERROR level.
- Update to OpenSSL 3.1.7 to mitigate security vulnerabilities.

Improvements

- Added support for binding console ports to a specific IP address when the TCP/IP provider is configured with multiple addresses. See the CONSOLE --console-ports option.

1.2.7 - 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.
- Update console footer documentation link,

- Update CAROOT file.
- Update libxml2 to version 2.12.5.
- Update openssl to version 3.1.5.

Improvements

- LWCCOM now supports the CONVERT API command, which allows conversion of API definitions to DDL from the TACL command line. Previously, this conversion could only be done using the Console.

1.2.6 - 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.

Improvements

- All console download links now send session tokens as HTTP headers instead of query params.

1.2.5 - 27 Mar 2023

Problems Corrected

- boolean fields are not handled correctly for XML
- XML add by example reports parseFloat error, when the example contains floating point data and the browser is IE11
- CLIENT may abend when message logging is configured and certain errors are returned from the collector serverclass
- 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

Improvements

- None

1.2.4 - 28 Sep 2022

Problems Corrected

- Swagger import generates invalid API & DDL when a JSON schema type contains only a primitive type definition
- DDL generator copies operation description characters that are invalid in DDL

- Schema editor find feature (CTRL-F) no longer works in v1.2.3
- OpenAPI import reports error when response content contains an empty object

Improvements

- Update PCRE version to mitigate vulnerabilities
- Add LWCCOM feature to validate licenses

1.2.3 - 17 May 2022

Problems Corrected

- XML deserializer ignores the charset found in the content-type header and returns an error if the content is not UTF-8
- CLIENT process logs error "ClientSocketCache discarded event ClientSocket.e.sendc"
- LWCCOM environment is invalid if the TACL _DEFAULTS define contains the system name
- SSL session closed error message does not show the request phase due to a formatting error
- CLIENT process reports an erro but does not shut down when the startup http-credentials file is invalid
- Console dashboard always shows CONSOLE running in CPU 0

Improvements

- Performance improvements in JSON de/serialization
- Add Event Message Reference to the documentation
- Update to OpenSSL 1.1.1n to mitigate CVE-2022-0778.
- Update libxml2 to v2.9.14 to mitigate security vulnerabilities.

1.2.2 - 28 Sep 2021

Problems Corrected

- API editor does not allow isArray, nillable, or isNull property on element
- Filesystem housekeeping process uses unnecessary read lock operations
- Setting isNull property to non-zero does not work on structures

Improvements

- Update to OpenSSL 1.1.1l

1.2.1 - 27 Mar 2021

Problems Corrected

[LC-982] - OpenAPI 3 import incorrectly imports arrays of objects.
[LC-986] - Message Logging meta data is only included when content spec includes "all"

Improvements

[LC-987] - Message logging sets IPM & HTTP lengths on all events, regardless of content selection

1.2.0 - 25 Jan 2021

Problems Corrected (since v1.1.2)

[LC-921] - --blob-files option is rejected if it contains more than 2 optional values
[LC-934] - CLIENT may abend when writing diagnostic log after request timeout
[LC-935] - CLIENT ignores socket connection reset while receiving response body on TLS connection
[LC-937] - Schema does not allow for definition of a single element array
[LC-942] - API Editor allows download and DDL generation when API schema has changed but has not been saved
[LC-947] - Diag log system info RVU is incorrect
[LC-949] - DDL generator does not treat ENVIRONMENT as COBOL reserved word
[LC-965] - LWCCOM pagesize command may report valid input as invalid
[LC-969] - OpenAPI 3 import produces schema with invalid type references
[LC-973] - Process logger may leak memory when debug logging is enabled
[LC-974] - Setting isSet: true does not work for JSON object serialization when the object is empty

New Features

Add support for TLS 1.3. See --tls-protocols
Add support for Message Logging
Allow LWCCOM users with read access to the filesystem to execute commands that don't modify the filesystem. See the ALLOW-READ-ACCESS option under the LWCCOM CONTROL FILESYSTEM command.
CONSOLE process now supports process log configuration using configuration files. See Using Configuration Files.
Add additional measure counters for diagnostic logging (lwc-diag-log) and message logging (lwc-msg-log). See Using Measure Counters.

1.3 Releases

1.3.1 - 15 July 2025

Problems Corrected

- Added SOAPam Compatible character substitution when UTF-8 to ISO-8859-1 conversion errors occur.
- Corrected high CLIENT process CPU utilization when SOAPam compatible statistics collector is enabled.
- Resolved XML deserialization error with BOM in payload and encoding in Content-Type.

- Corrected serialization error when a query param or form field name includes '.'.
- CLIENT now uses the content type charset when encoding the XML request payload.
- Console updated to allow import of Openapi 3.1 definitions.
- Certificate credentials file errors properly handled.
- Nested XML array deserialization no longer stops at first element.

Improvements

- Added legacy cipher support to LWCCOM.
- Update to OpenSSL 3.5.1 to support quantum cryptography and security vulnerabilities.
- Update Libxml2 to version 2.14.4 to address security vulnerabilities.

Product Licensing

LightWave Client 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_client
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 CONSOLE program subvolume
- The file "LICLWC" in the LightWave CONSOLE program subvolume
- The file "$SYSTEM.NUWAVE.LICLWC"

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 maintenance agreement expires. Once this expiration date has passed, the software will continue to run with full functionality (subject to the *expiration* date), 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.

Third Party Software Licenses

LightWave Client incorporates code from several open source projects. The licenses for these projects are included below:

- Ace (Ajax.org Cloud9 Editor)
- Ajv: Another JSON Schema Validator
- AngularJS
- Angular Material
- Apache Portable Runtime
- Apache Portable Runtime Utility Library
- Apache HTTP Server
- dlmalloc Memory allocator
- International Components for Unicode
- Jansson JSON Toolkit
- json-schema-generator
- cURL
- libxml2 XML Toolkit
- OpenSSL Toolkit
- Perl Compatible Regular Expressions

Ace (Ajax.org Cloud9 Editor)

Copyright (c) 2010, Ajax.org B.V.

https://github.com/ajaxorg/ace/blob/master/LICENSE

Ajv: Another JSON Schema Validator

Copyright (c) 2015 Evgeny Poberezkin

https://github.com/epoberezkin/ajv/blob/master/LICENSE

AngularJS

Copyright (c) 2010-2017 Google, Inc. http://angularjs.org

https://github.com/angular/angular.js/blob/master/LICENSE

Angular Material

Copyright (c) 2014-2016 Google, Inc. http://angularjs.org

https://github.com/angular/material/blob/master/LICENSE

Apache Portable Runtime

Copyright (c) 2011 The Apache Software Foundation.

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

Copyright (c) 2011 The Apache Software Foundation.

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

Copyright 2013 The Apache Software Foundation.

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

---

json-schema-generator

Copyright (c) 2014 krg7880

https://github.com/krg7880/json-schema-generator/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

Copyright (c) 1998-2011 The OpenSSL Project. All rights reserved.

Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)

http://www.openssl.org

http://www.openssl.org/source/license.html

Perl Compatible Regular Expressions

Copyright (c) 1997-2013 University of Cambridge

Copyright (c) 2009-2013 Zoltan Herczeg

Copyright (c) 2007-2012 Google Inc.

http://www.pcre.org/

http://www.pcre.org/licence.txt

Appendices

LightWave Client SOAPAM Compatibility Documentation

This Appendix describes the SOAPam Client Compatibility feature in LightWave Client.

Note: The SOAPam Client Compatibility feature is licensed separately from the base LightWave Client license. To acquire a license for this feature, please contact your NuWave account representative.

Introduction

LightWave Client SOAPAM Compatibility allows SOAPAM Client applications to use LightWave Client, without changing the SOAPam Client application. Using Client SOAPAM Compatibility, however, requires modification to operation procedures and configurations. These changes are documented elsewhere.

Startup

A LightWave Client process is created in the same manner as a SOAPAMCP process: either as a stand-alone process (with the `--standalone` option) started from a TACL prompt, or as a Pathway server process started by PATHMON.

Configuration

Whether running stand-alone or in a Pathway serverclass, the mechanisms used to configure LightWave Client are similar to those used to configure SOAPAM Client: command-line options, PARAMs, and configuration files. However, there are some differences.

API Definitions Instead of CDFs

LightWave Client is driven by an API definition. The LWC startup option `--api` specifies a filename that contains the API definition. The SOAPAMCP startup option `-cdf` is analogous.

Base URL Instead of Location

One key difference between the LightWave Client `--base-url` option and the SOAPAMCP `-location` option is that the `--base-url` option does not include the operational path. That value is specified in the API.

For example:

LightWave Client

`--base-url`	http://soapam-echotest.demo.nuwavetech.com
API operation path	/services/echostring/echostring

SOAPAMCP

`-location`	https://soapam-echotest.demo.nuwavetech.com/services/echostring/echostring

Another difference between the LWC `--base-url` option and the SOAPAMCP `-location` option is that the `--base-url` option is required because the LightWave Client API definition does not contain URLs as the SOAPAMCP CDF does.

Diagnostic Logs Instead of Message Dumps

LightWave Client Diagnostic Logs are analogous to SOAPAM Message Dumps. The content is different to reflect the differences between a RESTful API call and a SOAP/XML web service invocation.

Operationally, diagnostic logs are separate files for each API call, and they are always EDIT-type files. The option to write all diagnostic log entries to a single file is not available in LWC.

Additionally, diagnostic log filters are specified via the `--diag-log` option instead of separate options (typically contained in an options file).

Log File

LightWave Client logging is configured with the single `--log` option, whether values are provided on the command-line or in a configuration file.

Converting Credentials Files

SOAPAMCP credentials files created with the *makecf* utility are not valid for LightWave Client. Credentials files must be recreated with the LightWave Client LWCOM utility's CREATE CREDENTIALS command.

```
CREATE CREDENTIALS <file-name> [ ! ]
  [ , CREDENTIALS <credentials-string> ]
  [ , FROM-FILE <file-name> ]
  [ , TARGET <file-name> ]
  <file-name> [ ! ]
```

File to contain the stored credentials. Append `!` to the name to overwrite an existing file with the same name.

`CREDENTIALS <credentials-string>`

The credentials to be encoded, e.g. <username>':'<password>.

`FROM-FILE <file-name>`

A file containing unencoded credentials to be encoded.

`TARGET <file-name>`

The file name to be encoded with the stored credentials. If not specified, it will be the same name as the output file name.

Converting CDFs

The LightWave Client command interpreter (LWCCOM) has been enhanced to convert SOAPAMCP CDFs to LWC APIs and to Data Definition Language (DDL) source files.

CDF to API

```
run lwccom convert cdf <cdf-filename>, format api, file <api-
filename> [ ! ]
```

Where:

`<cdf-filename>`	The filename of the CDF to convert.
`format api`	Specifies the output file is “LWC API.”
`<api-filename>`	Specifies the name of the LWC API file. An optional exclamation point after the filename indicates any existing output file should be replaced.

CDF to DDL

Convert a CDF to a DDL source file with the following LWCCOM command:

```
run lwccom convert cdf <cdf-filename>, format ddl, file <ddl-filename> [ ! ]
```

Where:

`<cdf-filename>`	The filename of the CDF to convert.
`format ddl`	Specifies the output file is “NonStop DDL source.”
`<ddl-filename>`	Specifies the name of the DDL source file. An optional exclamation point after the filename indicates any existing output file should be replaced.

NOTE: The LWCCOM convert CDF to DDL and export API to DDL commands generate two different versions of application structures in the DDL source. The convert CDF to DDL command generates DDL source in a format suitable for exchanging messages with LightWave Client running in SOAPam Compatibility mode. The export API to DDL generates DDL source for use with LightWave Client running in “native” mode.

Command-line Option Cross References

Note: A full description of the options in these tables is beyond the scope of this document. Please see relevant product documentation for details.

SOAPAMCP to LightWave Client

The following table identifies SOAPAM Client command-line options and their LightWave Client equivalents. If the command is unavailable or not applicable, N/A appears in the column.

SOAPAM Client	LightWave Client	Notes
`@<command-file>`	*same*	
`-authheader`	*same*	
`-backupcpu`	*same*	
`-cdf`	`--api`	Similar purpose but specify an API definition filename.
`-disablehttpkeepalive`	N/A	
`-enabletcpkeepalive`	N/A	
`-help`	N/A	
`-httpauth`	`--http-credentials`	
`-httpkeepalivetimeout`	`--http-keepalive-timeout`	
`-httppreauth`	`--http-credentials`	The ‘preauth’ keyword is appended to the LWC option to enable preauth.

SOAPAM Client	LightWave Client	Notes
`-httpproxy`	`--http-proxy-host`	
`-httpproxyauth`	`--http-proxy-credentials`	
`-httprequesttimeout`	`--http-request-timeout`	*The LWC option value is expressed in **milliseconds**, not seconds as with SOAPAMCP.*
`-ignoreclose`	`--ignore-close` `--standalone`	Either LWC option is acceptable.
`-licensefile`	`--license`	
`-licensekey`	N/A	
`-location`	`--base-url`	The option value is different.
`-log`	`--log`	For specifying options on the command-line.
`-logcfg`	`--log +<logopt file>`	For specifying options in a file. The time= option is not supported.
`-logtime`	N/A	All log times are in LCT (when using text format).

SOAPAM Client	LightWave Client	Notes
`-messagedump`	`--diag-log`	Options are different for LWC diagnostic logging. The content filter in a diagnostic log configuration file is analogous to the SOAPAM option on the command-line.
`-messagedumpcfg`	`--diag-log +<diaglog opt file>`	For specifying diagnostic logging options in a file.
`-messagedumpfile`	N/A	Each diagnostic is written to a separate file.
`-messagedumpfiletype`	N/A	Diagnostic logs are always EDIT-type files.
`-messagedumpfilter`	`--diag-log +<diaglog opt file>`	Use the replyCodeFilter and/or httpStatusFilter options in a diagnostic log configuration file.
`-messagedumpsubvol`	`--diag-log`	Either specify the subvol= option on the LWC command-line or use it in a diagnostic log configuration file.
`-pathmonpattern`	N/A	
`-resolveonconnect`	N/A	
`-servicedefaultencoding`	`--default-encoding`	

SOAPAM Client	LightWave Client	Notes
`-sslcalocalfile`	`--ca-local-certs`	
`-sslcarootfile`	`--ca-root-certs`	
`-sslclientcert`	`--client-cert`	

`-ssl-disable-tlsv1.0`
`-ssl-disable-tlsv1.1`
`-ssl-disable-tlsv1.2`
`-ssl-disable-tlsv1.3`
	`--tls-disable-v1.0`
	`--tls-disable-v1.1`
	`--tls-protocols`	

`-sslnoverify`	`--cert-no-verify`	
`-stats`	N/A	Statistics configuration in LWC is available only via a configuration file.
`-statscfg`	same	
`-switchontimeout`	N/A	Functionality not supported.
`-tcpip`	`--tcpip-process`	
`-tcpipalt`	N/A	Functionality not supported.

LightWave Client to SOAPAMCP

The following table identifies LightWave Client command-line options and their SOAPAM Client equivalents. If the command is unavailable or not applicable, N/A appears in the column.

LightWave Client	SOAPAM Client	Notes
`@<command-file>`	*same*	Same purpose but the options in the file would appropriate to the product.
`--api`	`-cdf`	
`--api-param-<param-name>`	N/A	Specifies an API specific parameter name and its value. E.g., the option `--api-param-queueme` would specify the value for an API parameter called *queueme*.
`--auth`	N/A	Specifies the name of an authorization file used in request authentication and signing.
`--base-url`	`-location`	The option value is different; the LWC value does not include the operation's path.
`--blob-files`	N/A	
`--ca-local-certs`	`-sslcalocalfile`	
`--ca-root-certs`	`-sslcarootfile`	
`--cert-no-verify`	`-sslnoverify`	

LightWave Client	SOAPAM Client	Notes
`--client-cert`	`-sslclientcert`	
`--default-encoding`	`-servicedefaultencoding`	

`--diag-log`	`-messagedump`
	`-messagedumpcfg`
	`-messagedumpfile`
	`-messagedumpfiletype`
	`-messagedumpfilter`
	`-messagedumpsubvol`
		The LWC `--diag-log` option takes the place of these SOAPAMCP options.

`--filesystem`	`-vfs`	
`--http-credentials`	`-httpauth`	
`--http-keepalive-max`	N/A	
`--http-keepalive-timeout`	`-httpkeepalivetimeout`	
`--http-proxy-host`	`-httpproxy`	

LightWave Client	SOAPAM Client	Notes
`--http-proxy-credentials`	`-httpproxyauth`	
`--http-request-timeout`	`-httprequesttimeout`	*The LWC option value is expressed in **milliseconds**, not seconds as with SOAPAMCP. E.g.: 1000 = 1 second.*
`--ignore-close`	`-ignoreclose`	
`--license`	`-license` `-licensefile`	
`--log`	`-log` `-log-cfg`	The LWC `--log` option takes the place of both SOAPAMCP options.
`--monitor`	N/A	
`--msg-log`	N/A	
`--soapamcp-mode`	N/A	Enables the SOAPAMCP compatibility mode of LWC.
`--standalone`	`-ignoreclose`	
`--statscfg`	same	
`--string-padding`	N/A	
`--sts-max-age`	N/A	

LightWave Client	SOAPAM Client	Notes
`--tcpip-bind-addr`	N/A	
`--tcpip-process`	`-tcpip`	
`--tls-cipher-list`	N/A	
`--tls-options`	N/A	

`--tls-disable-v1.0`
`--tls-disable-v1.1`
`--tls-protocols`
	`-ssl-disable-tlsv1.0`
	`-ssl-disable-tlsv1.1`
		The LWC `--tls` -protocols option specifies which TLS versions to make available.

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 Client product version:

```
> VPROC [<installation-subvolume>.]CLIENT
```

Copy and paste the VPROC output to your support request.

Include a Copy of the API Definition

If your issue concerns a specific API that you are using, attach a copy of the API to your Support Request. You can obtain a copy of your API by using the Console. Navigate to the API definition at issue. In the API view toolbar, click the (down-arrow icon) to export the API to a file. Alternatively, you can use the LWCCOM *EXPORT API* command to extract the API definition to a file on your NonStop system, then download the file to your personal computer.

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.

Client Application Source Code

If your problem relates to a specific API method invocation, consider including the relevant source code from your client application with your Support Request. That is, the source code that constructs and sends the request and receives and examines the response.

LightWave Client Documentation

Release 1.3.1

Document

Classification: Proprietary and Confidential

Type: Customer Documentation

Author: NuWave Technologies, Inc.

Document Version: 1.3.1

Release Date: 15 July 2025

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

NuWave