Working with User-Defined APIs

Overview

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

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

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

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

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

RESTful API Design Pattern

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

/my-api/v1/employee

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

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

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

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

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

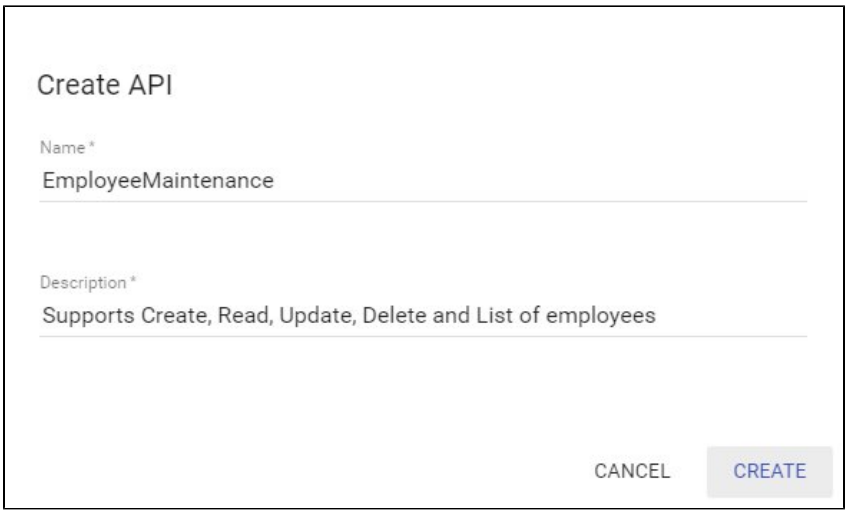
DELETE - **D**eletes the employee with id {employee-id}

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

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

Create an API

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

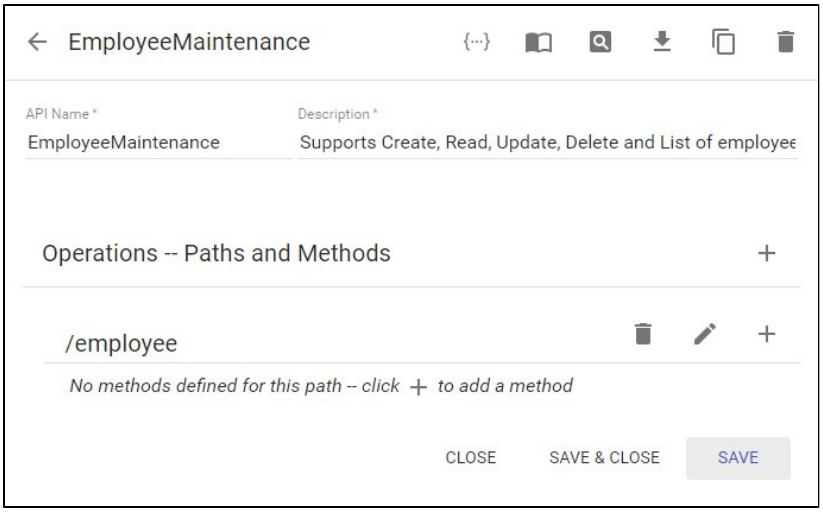


Add a Path

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

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

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

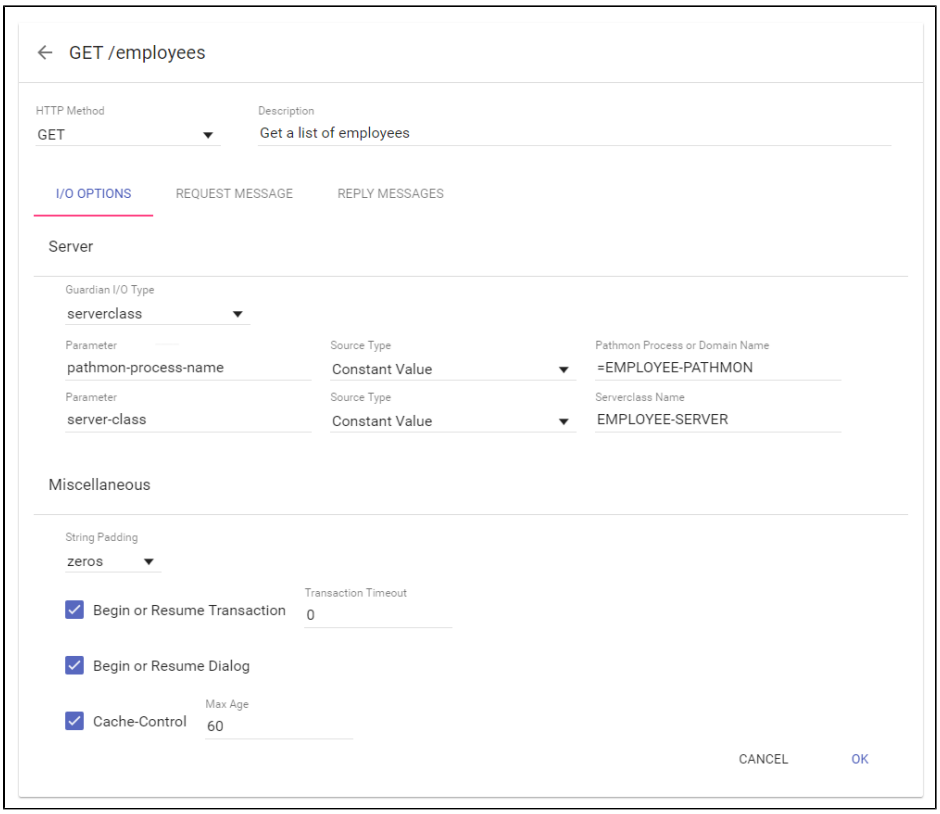


Path specifications do not include "query parameters", such as 'details' in the URI /employee/{employee-id}?details=1. Rather, query parameters are part of a method's request message field mappings.

Add a Method for a Path

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

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



I/O Options

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

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

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

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

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

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

Miscellaneous

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

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

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

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

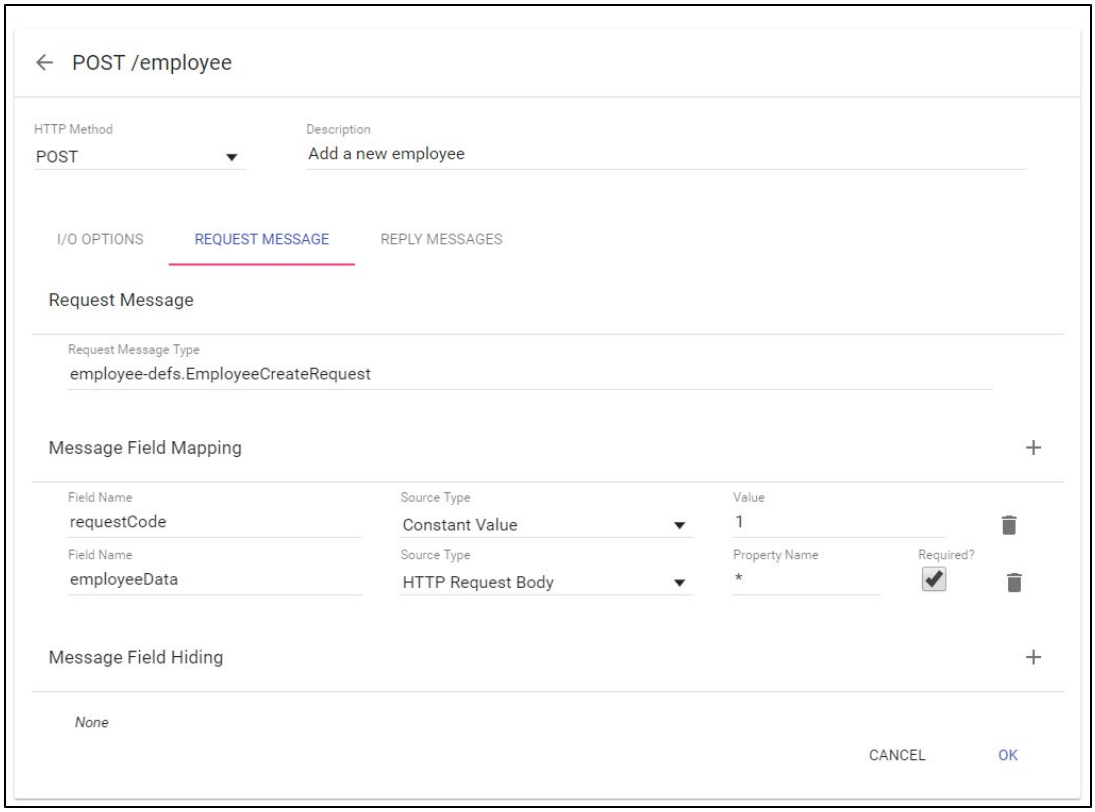
Request Message

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

Message Field Mapping

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

<table>
<tr><th>Source Type</th><th>You enter / select</th></tr>
<tr><td>Constant Value</td><td>a string, numeric value, or substitution variable</td></tr>
<tr><td>URI Path Component</td><td>a {path-variable} you specified in the path for this method</td></tr>
<tr><td>URI Query Parameter</td><td>the name of a query parameter that will be supplied in the request URI</td></tr>
<tr><td>HTTP Request Body</td><td>the name element that will be supplied in the request body, or "*" for the entire request body</td></tr>
<tr><td>HTTP Request Header</td><td>the name of the HTTP header supplied with the request</td></tr>
<tr><td>HTTP Request Element</td><td>The name of an element of the HTTP request:<table>
<tr><td>URI</td><td>The URI of the request, e.g., /employee/8213912?history=true</td></tr>
<tr><td>method</td><td>The HTTP method (verb) of the request, e.g., GET, PUT, ...</td></tr>
<tr><td>username</td><td>The authenticated username of the caller</td></tr>
</table></td></tr>
<tr><td>HTTP Form Field</td><td>when the request body is 'form-encoded', the name of a form field</td></tr>
</table>



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

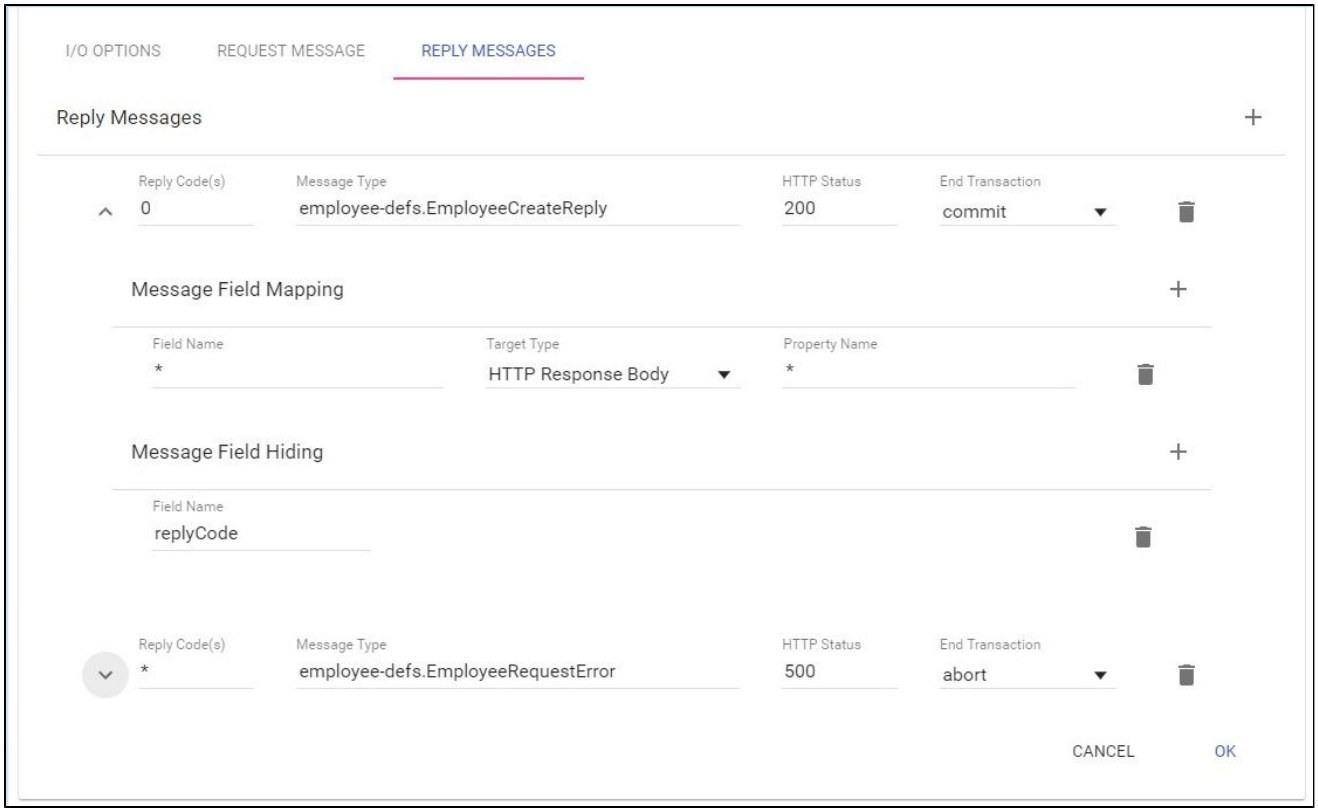
Message field mappings of some Source Types may be specified as 'required' by checking the Required checkbox. If a required mapping is not supplied in the request by the client, LightWave not send the request to the server and will return an error to the client.

Message Field Hiding

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

Reply Messages

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



Reply Code(s)

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

Message Type

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

HTTP Status

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

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

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

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

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

Transaction State

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

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

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

Dialog State

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

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

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

Message Field Mapping

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

Message Field Hiding

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

Using Substitution Variables

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

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