Article
Ward De Backer · Jun 23, 2016 5m read

Returning custom responses from Web Services and REST services using Node.js and EWD 3

As Rob explained in an earlier post, Caché's Node.js interface allows you to create Web Services and REST Services using the very modular EWD 3 framework.

These services by default return a JSON response with Content-Type: application/json and the response body contains the JSON you return using the finished() method, so:

finished({ test: 'test response' });

returns

{ "test": "test response" }

with a HTTP content-type of application/json

For other external services and tools like e.g. reporting engines or EDI, you will need to return responses formatted to the specs of these services and also with different content-types like XML. However, because ewd-qoper8-express does all the necessary plumbing work for you by default to return and send the JSON response, you will need to modify and format the response in another format as the specs of the external service requires before it is sent out. With the ewd-qoper8-express module it is possible now to modify the HTTP response to your own needs.

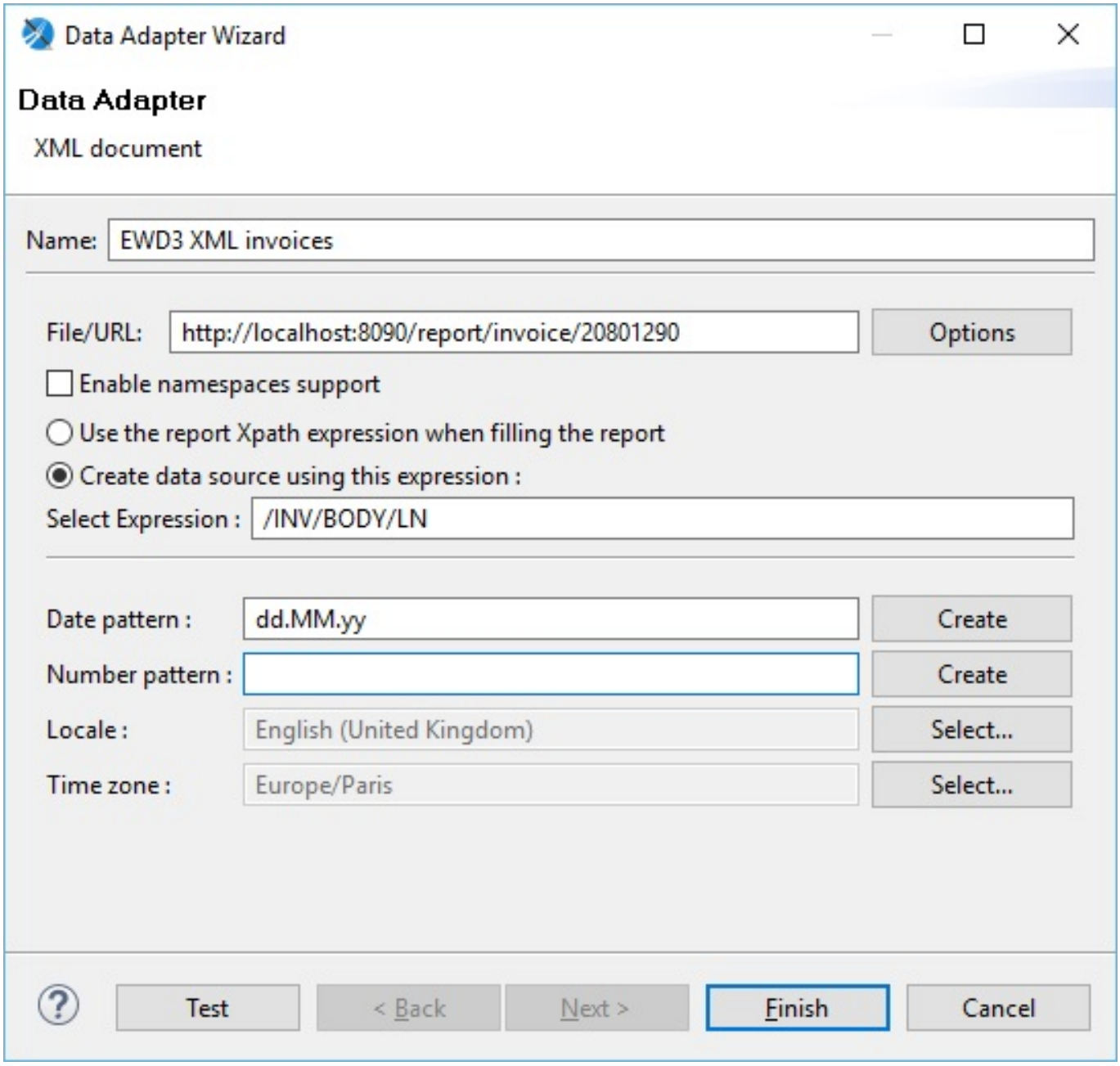
As described in Rob's previous post how to create services, you know the Express module provides you with a very clear pattern to define each service in your ewd-xpress.js startup file using:

```
var config = {
  managementPassword: 'keepThisSecret!',
  serverName: 'My EWD 3 Server',
  port: 8080,
  poolSize: 1,
  database: {
    type: 'cache'
  }
};

var ewdXpress = require('ewd-xpress').master;
var xp = ewdXpress.intercept();
xp.app.use('/test', xp.qx.router());
xp.app.use('/report', xp.qx.router());

ewdXpress.start(config);
```

This will by default return a HTTP response in JSON for the service <test> and <report>.

As an example, if you want to use an external reporting tool like JasperReports, if you are using Caché classes and SQL you can use its JBDC Data Connection to provide the data for your reports, but if your data is in globals or you

want to use the document-oriented approach, its XML data adapter can be a better choice to provide the data for your report. In this case, you'll need to create a REST service that returns the data for the report in XML format.



To customize the response and return it in XML instead of JSON, we can use the same pattern as Express uses for adding custom middleware. The key to tell the ewd-qoper8-express module you will return a custom response yourself, is to add a {nextCallback: true} option to the router function call in xp.app.use() and to add your own callback to the callback chain which will send out the response according to the format needed. E.g. when you need to return a reponse to an XML data adapter for JasperReports, your ewd-xpress.js startup file becomes:

```
var config = {
  managementPassword: 'keepThisSecret!',
  serverName: 'My EWD 3 Server',
  port: 8080,
  poolSize: 1,
  database: {
    type: 'cache'
  }
};
```

```
var ewdXpress = require('ewd-xpress').master;
var xp = ewdXpress.intercept();
var js2xmlparser = require('js2xmlparser');

xp.app.use('/report', xp.qx.router({ nextCallback: true }), function(req, res) {
 var message = res.locals.message;

 console.log('##### converting response to XML ... #####');
 console.dir(message, {depth:6});
 res.set('Content-Type', 'application/xml');
 if (message.error)
  res.send(js2xmlparser('error', message));
 else
  res.send(js2xmlparser(message.json.root || 'xmlRoot', message.json.data || {}));
});
```

This works exactly the same as in the previous example returning a JSON response, the handlers in your report.js module will be called for each report type, but we will now convert the JSON to XML afterwards using the next callback in the express chain. In this case, you need to send out the response in your code as the default response handling/sending logic is bypassed in this case. The JSON response from your application module is returned in res.locals.message now and you can modify it now before you send it out.

You'll notice that another npm module (js2xmlparser) in addition to the Express module is used here to convert the JSON response returned by your application module (report.js in this example) to XML. This is one of the most powerful features of Node.js because you can use all ready to use modules (currently 250.000!) in your Caché applications.

Your report.js application module can return now a response like this (see JasperReports example):

```
{
  json: {
    data: {
      'category': [
        {
          '@': {
            'name': 'home'
          },
          'person': [
            {
              '@': {
                'id': 1
              },
              'lastname': 'Davolio',
              'firstname': 'Nancy'
            },
            ...
          ]
        },
        ...
      ]
    },
    root: 'addressbook'
  },
  error: ''
}
```

As you'll notice, the '@' is used to return attributes of an XML tag. Finally, to return the response in XML, you need to set the Content-Type to application/xml and the js2xmlparser module converts your JSON response to XML:

```
<addressbook>
  <category name="home">
    <person id="1">
      <lastname>Davolio</lastname>
      <firstname>Nancy</firstname>
    </person>
    ...
  </category>
  ...
</addressbook>
```

Another example is creating a REST service for creating documents used for EDI. As these documents need to be exchanged with other parties, they're in XML format.

One more tip: if you are debugging your application module using e.g. Chrome's Advanced REST client and you want to return/see the JSON response from your application module, you can change {nextCallback:true} to {nextCallback:false} and your extra callback function will not be called (and not convert to XML). Useful to inspect the JSON if the XML is not as expected.

You can customise now the HTTP responses completely to your own needs and you can use the standard patterns provided by the Express module!

#Caché #JSON #Node.js #REST API #SOAP

Source URL:https://community.intersystems.com/post/returning-custom-responses-web-services-and-rest-services-using-nodejs-and-ewd-3