

How do I debug my API?

Debugging an API is essentially no different than debugging any stored procedure. Therefore, a developer can simply put a debug statement in the code. However, this might not be desirable if the web APIs are being accessed by regular users. This article will provide a couple of techniques for **debugging and analyzing APIs** with minimal impact toward other users.

First Things First

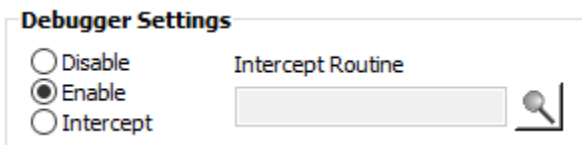
In order to interact with the visual debugger there are a couple of configuration steps that are required.

1. Run the Engine Server in "Debug" Mode

The first step is to make sure you are running the Engine Server from a command prompt (aka "debug" mode). In our [Testing for Success](#) article, we provide a sample screen shot of how to launch the Engine Server from a command prompt. It is important to remember that if the Engine Server is also configured to run as a Windows Service then this will need to be stopped in order for the Engine Server in the command prompt to run successfully. Further information can be found in the official *103-966 OpenInsight OEngineServer Configuration.pdf* document.

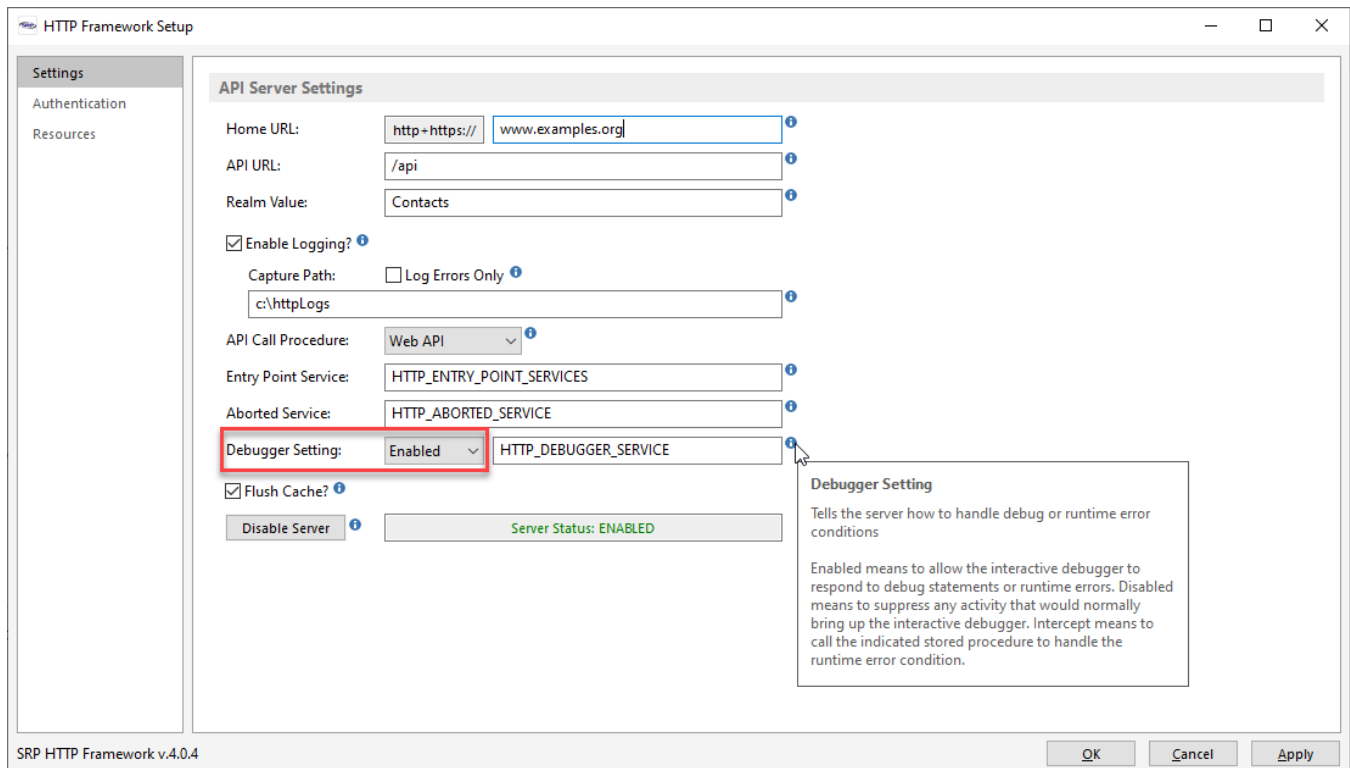
2. Enable the Debugger

The second step is to make sure the engines handling the HTTP request are set to enable the visual Debugger. By default, headless engines will use the Debugger Settings from the Database Manager:



The image shows a 'Debugger Settings' dialog box. It has three radio buttons: 'Disable', 'Enable' (which is selected), and 'Intercept'. To the right of the radio buttons is a label 'Intercept Routine' and a text input field. There is also a magnifying glass icon on the right side of the dialog.

However, there might be times when the production database wants to avoid the *Enable* setting but the API developer needs to use the visual Debugger. For reasons like this, the [HTTP Framework Setup](#) form provides the same options as the Database Manager but this setting will only affect those engines handling HTTP requests:



The image shows the 'HTTP Framework Setup' dialog box. It has a sidebar with 'Settings', 'Authentication', and 'Resources'. The main area is titled 'API Server Settings' and contains several fields: 'Home URL' (http+https:// www.examples.org), 'API URL' (/api), 'Realm Value' (Contacts), 'Enable Logging?' (checked), 'Capture Path' (c:\httpLogs), 'API Call Procedure' (Web API), 'Entry Point Service' (HTTP_ENTRY_POINT_SERVICES), 'Aborted Service' (HTTP_ABORTED_SERVICE), 'Debugger Setting' (Enabled), and 'Flush Cache?' (checked). There is a 'Disable Server' button and a 'Server Status: ENABLED' indicator. A tooltip for the 'Debugger Setting' dropdown explains that 'Enabled' means to allow the interactive debugger to respond to debug statements or runtime errors, 'Disabled' means to suppress any activity that would normally bring up the interactive debugger, and 'Intercept' means to call the indicated stored procedure to handle the runtime error condition. The bottom of the dialog has 'OK', 'Cancel', and 'Apply' buttons.

Triggering the Debugger

The most direct way to debug an API is to debug the code. However, as noted above, simply embedding a debug statement in the API code would interrupt all requests. So there needs to be a way to safely trigger the debugger. In traditional OpenInsight development, developers could check for unique conditions such as the values in the @USERNAME or @STATION system variables. Since all HTTP requests will always be handled by the same username and station, this method isn't useful.

One way to introduce unique conditions is to add [custom parameters](#) to the API request. API testing tools like Postman make it very easy to add custom parameters. We recommend updating the *AdditionalValues* string from the *OECGI4* registry key to include **HTTP_DEBUG**. Doing so will tell the OECGI to pass through a request header with the name **Debug**. You could, of course, create any name (or *names*) you want.

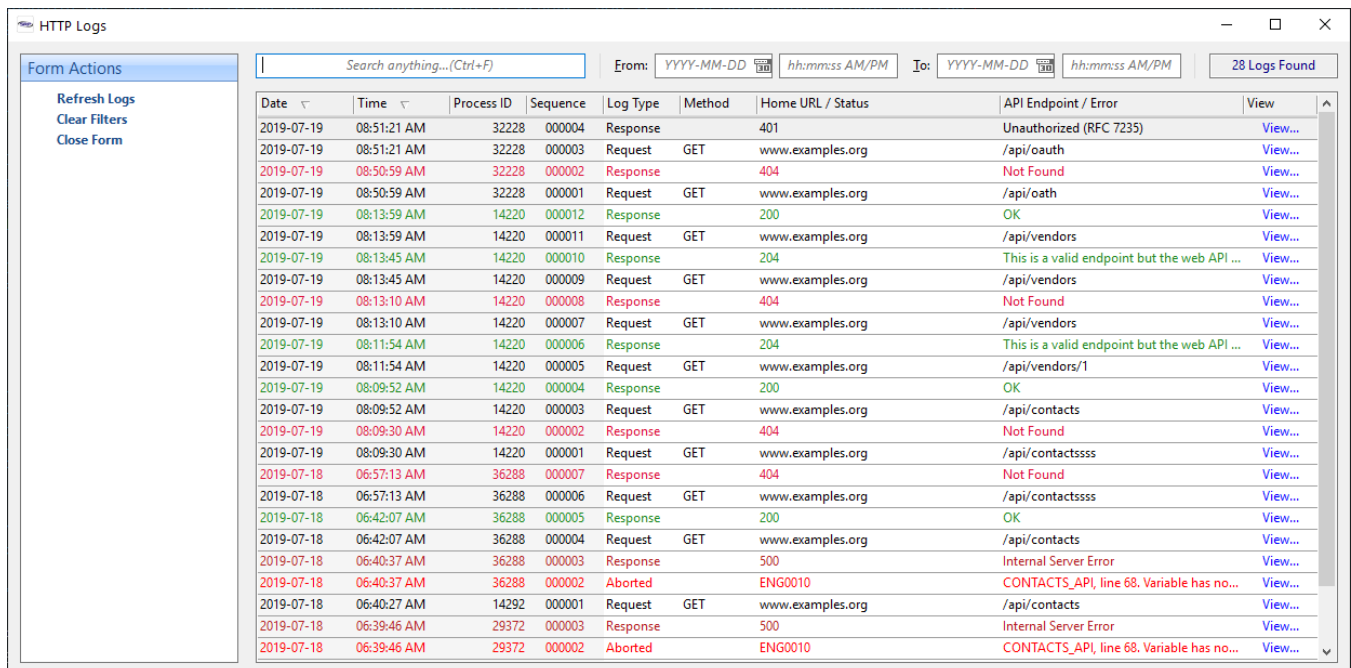
The **HTTP_MCP** controller routine already has code that checks for a value assigned to the **Debug** request header:

```
// Engage the debugger if requested.  
If HTTP_Services('GetRequestHeaderField', 'Debug') then Debug
```

This same code could be copied anywhere and adapted as needed (e.g., change the name of the header field or check for a specific field value). APIs can now be easily debugged without impacting regular API calls.

Examining the Log Files

While not the same as using the Debugger in an interactive manner, the log files can provide an abundance of useful information. For instance, the entire OECGI request array is maintained in each *Request* log. This is structured so it is easy to review (read: delimiters are replaced with human readable formatting). The entire response (with some useful metadata) is maintained in each *Response* log. Furthermore, depending upon how your *Debugger Setting* is configured, runtime errors will appear in a *Debugger*, *Aborted*, or *GetStatus* log. If you already know some details about a specific request that you need to investigate (e.g., status code, date/time range, endpoint, etc.) you can quickly find the relevant logs using the **HTTP Logs** utility (just launch the *NDW_HTTP_LOGS* form):



Date	Time	Process ID	Sequence	Log Type	Method	Home URL / Status	API Endpoint / Error	View
2019-07-19	08:51:21 AM	32228	000004	Response		401	Unauthorized (RFC 7235)	View...
2019-07-19	08:51:21 AM	32228	000003	Request	GET	www.examples.org	/api/oauth	View...
2019-07-19	08:50:59 AM	32228	000002	Response		404	Not Found	View...
2019-07-19	08:50:59 AM	32228	000001	Request	GET	www.examples.org	/api/oauth	View...
2019-07-19	08:13:59 AM	14220	000012	Response		200	OK	View...
2019-07-19	08:13:59 AM	14220	000011	Request	GET	www.examples.org	/api/vendors	View...
2019-07-19	08:13:45 AM	14220	000010	Response		204	This is a valid endpoint but the web API ...	View...
2019-07-19	08:13:45 AM	14220	000009	Request	GET	www.examples.org	/api/vendors	View...
2019-07-19	08:13:10 AM	14220	000008	Response		404	Not Found	View...
2019-07-19	08:13:10 AM	14220	000007	Request	GET	www.examples.org	/api/vendors	View...
2019-07-19	08:11:54 AM	14220	000006	Response		204	This is a valid endpoint but the web API ...	View...
2019-07-19	08:11:54 AM	14220	000005	Request	GET	www.examples.org	/api/vendors/1	View...
2019-07-19	08:09:52 AM	14220	000004	Response		200	OK	View...
2019-07-19	08:09:52 AM	14220	000003	Request	GET	www.examples.org	/api/contacts	View...
2019-07-19	08:09:30 AM	14220	000002	Response		404	Not Found	View...
2019-07-19	08:09:30 AM	14220	000001	Request	GET	www.examples.org	/api/contactsssss	View...
2019-07-18	06:57:13 AM	36288	000007	Response		404	Not Found	View...
2019-07-18	06:57:13 AM	36288	000006	Request	GET	www.examples.org	/api/contactsssss	View...
2019-07-18	06:42:07 AM	36288	000005	Response		200	OK	View...
2019-07-18	06:42:07 AM	36288	000004	Request	GET	www.examples.org	/api/contacts	View...
2019-07-18	06:40:37 AM	36288	000003	Response		500	Internal Server Error	View...
2019-07-18	06:40:37 AM	36288	000002	Aborted		ENG0010	CONTACTS_API, line 68. Variable has no...	View...
2019-07-18	06:40:27 AM	14292	000001	Request	GET	www.examples.org	/api/contacts	View...
2019-07-18	06:39:46 AM	29372	000003	Response		500	Internal Server Error	View...
2019-07-18	06:39:46 AM	29372	000002	Aborted		ENG0010	CONTACTS_API, line 68. Variable has no...	View...

Careful study of the logs files can often avoid the need to debug code. In the above screen shot, sample runtime errors can be seen in the *Aborted* log types. Unexpected conditions that are not due to runtime errors (like 4xx responses) can also be easily inspected with the relevant logs.