# Sending debug logs to the server in Pas2JS

Michaël Van Canneyt

October 23, 2023

**Abstract**

In this article we show how to use a ready-to-use mechanism for sending debug logs from a pas2js program to a HTTP server application written in Free Pascal.

## 1 Introduction

In an ideal world, the application runs smoothly, and all eventualities during execution of a program are handled gracefully. If this were so, debug logging and unexpected error handling can be stripped once the program is ready for shipment.

In reality, programs or their execution environment are not perfect, and users do unexpected and unforeseen things: for these two reasons, often you must still have debugging logs in shipped applications.

In the browser, the all-time pascal 'WriteLn()' statement can be used to write to the browser console. If so desired, the result can be shown in the HTML. But as soon as the user closes the browser window, this information is lost. For most users, finding and transmitting the information in the browser console at the request of a support team is a difficult, not to say impossible task.

Therefor a better solution to gather debug info is to send it directly to the webserver, where the logs can be examined at once or saved to be examined later on.

In this article, we demonstrate a mechanism for transmitting such debug information. This mechanism is included by default in Free Pascal and Pas2js: `debugcapture`.

## 2 Architecture

The debug capture functionality - naturally - consists of 2 parts: one part is included in fcl-web, the other is part of pas2js, and is used in a pas2js client program.

The `fpDebugCaptureSvc` unit is part of Free Pascal's fpweb package for making HTTP server applications: You can include it in a HTTP server program and with a single line of code activate it. It is included by the `simpleserver` application by default. The compileserver program included in pas2js also provides this functionality. The functionality is disabled by default, the `-u` command-line switch must be provided to enable the debug capture: if no extra argument is given the captured info is printed on the console. If an extra argument is given to the `-u` option , it is interpreted as a filename in which to save the output. The URL for the service is `/debugcapture/` by default.

The client part is contained in the `debugcapture` unit, part of Pas2JS. It contains a simple client component that sends the output to a configurable URL.

We'll demonstrate the use of both sides in the rest of this article.

## 3 The server part: `TDebugCaptureService`

The `fpDebugCaptureSvc` unit contains a `TDebugCaptureService` component. It can be used to handle one or more HTTP routes. It can log to console or file by default, but additional backends can be registered.

This component has the following declaration:

```
TDebugCaptureHandler =
   Procedure (aSender : TObject; aCapture : TJSONData) of object;
TDebugCaptureLogHandler =
   Procedure (EventType : TEventType; const Msg : String) of object;

TDebugCaptureService = class(TComponent)
  class Property Instance : TDebugCaptureService;
  class function JSONDataToString(aJSON: TJSONData): TJSONStringType;
  Procedure HandleRequest(ARequest: TRequest;
                          AResponse: TResponse);
  Procedure RegisterHandler(const aName : String;
                            aHandler: TDebugCaptureHandler);
  Procedure UnregisterHandler(const aName : String);
  Property LogFileName : string;
  Property LogToConsole : Boolean;
  Property CaptureToErrorLog : Boolean;
  Property OnLog : TDebugCaptureLogHandler;
  Property CORS : TCORSSupport;
end;
```

The following methods exist:

**HandleRequest** This is the entry point of the service: the signature of this method is such that it can be used as the handler of a route in the fpWeb server's HTTP router.

**RegisterHandler** You can add as many handlers for a debug capture request as you want. You register a callback `aHandler` with a (unique) name `aName`. The name is used in log messages when appropriate, and can be used to unregister the handler.

**UnregisterHandler** can be used to unregister a handler with given name from the list of debug capture handlers.

**JSONDataToString** this class method can be used to convert the JSON payload to a string. It will take case of special cases such as null or objects.

The following properties can be used:

**Instance** This is a global instance of the component. This can be used for quickly setting up an instance of the debug capture service.

**LogFileName** When set to a non-empty, logging captured debug output to file is enabled.

**LogToConsole** When set to `True`a non-empty, captured debug output is sent to the console.

**CaptureToErrorLog** When set to `True`, output is sent to the OnLog log handler together with error messages from the component.

**OnLog** This event is used to log error messages from the component: when an error happens during writing of debug output to one of the handlers, it is logged using this event. If `CaptureToErrorLog` is set to true, all captured debug output is also sent to this event.

**CORS** This can be configured to handle CORS preflight requests, enabling you to run the debug capture service on a different URL from where your application is served. Make sure you configure CORS correctly if you enable it, it is a bad idea to allow all possible domains to use this service.

The 3 standard logging mechanisms (file, console, errorlog) use the `RegisterHandler` and `UnregisterHandler` calls, so they are called in the same manner as your own handlers. Any errors when writing to file or console will therefor also be reported using the standard log mechanism.

The use of this component is very simple. The following little program is a complete webserver that also has the debugcapture output. It overrides 2 methods of the standard `TCustomHTTPApplication` class to provide logging and to configure the server:

```
program demosvr;

uses
  custhttpapp,  sysutils, Classes, jsonparser, fpjson, httproute,
  httpdefs, fpmimetypes, fpwebfile, fpwebproxy, fpdebugcapturesvc;

Type
  { THTTPApplication }

  THTTPApplication = Class(TCustomHTTPApplication)
  private
    procedure HandleCaptureOutput(aSender: TObject; aCapture: TJSONData);
  published
    procedure DoLog(EventType: TEventType; const Msg: String); override;
    Procedure Initialize; override;
  end;

procedure THTTPApplication.DoLog(EventType: TEventType; const Msg: String);
begin
  Writeln(FormatDateTime('yyyy-mm-dd hh:nn:ss.zzz',Now),' [',EventType,'] ',Msg)
end;
```

Here we have done nothing yet except define our class and implement logging.

The override of the `DoRun` method is where the magic happens: the standard instance of the `TDebugCaptureService` is used to provide the debug capture functionality. It is configured to send the debug output to the console and to a file called `debug.log` by setting the `LogToConsole` and `LogFileName` properties:

```
procedure THTTPApplication.Initialize;
```

```
var
  aBaseDir : String;
  Svc : TDebugCaptureService;

begin
  Port:=8080;
  Svc:=TDebugCaptureService.Instance;
  Svc.OnLog:=@DoLog;
  Svc.LogFileName:='debug.log';
  Svc.RegisterHandler('log',@HandleCaptureOutput);
  HTTPRouter.RegisterRoute('/debugcapture',rmPost,@Svc.HandleRequest,False);
  aBaseDir:=IncludeTrailingPathDelimiter(GetCurrentDir);
  TSimpleFileModule.RegisterDefaultRoute;
  TSimpleFileModule.BaseDir:=aBaseDir;
  TSimpleFileModule.OnLog:=@Log;
  TSimpleFileModule.IndexPageName:='index.html';
  MimeTypes.LoadKnownTypes;
  inherited;
end;
```

After registering the `/debugcapture` route, the standard `TSimpleFileModule` component is used to provide standard HTTP file serving from the current directory. Note that we will not use the standard mechanism to log to console, instead, we implement our own handler: `HandleCaptureOutput`, which we register with the name `Log`. (the names for the internal logging mechanisms all start with $, do not use this character in your own handlers)

The `HandleCaptureOutput` method uses the `JSONDataToString` class method to create a string and logs it using the standard `DoLog` method of the application class.

```
procedure THTTPApplication.HandleCaptureOutput(aSender: TObject; aCapture: TJSONData);
begin
  DoLog(etDebug,TDebugCaptureService.JSONDataToString(aCapture));
end;
```

As a result, the debug info and the info about served pages is displayed in the same uniform manner.

With this, the application class is ready, all that needs to be done is to start it:

```
Var
  Application : THTTPApplication;

begin
  Application:=THTTPApplication.Create(Nil);
  Application.Initialize;
  Application.Run;
  Application.Free;
end.
```

And so, with 20 lines of code, we have created a HTTP server that also acts as a receiver of debug log info.

# 4 The client part: TDebugCaptureClient

In Pas2JS, the debugcapture unit provides the TDebugCaptureClient component.

```
TDebugCaptureClient = class(TComponent)
Public
  Class property Instance : TDebugCaptureClient Read _Instance;
  Procedure Capture(const aLine : String; NewLine : Boolean = True); virtual;
  Procedure Flush;
  Procedure SetConsoleHook;
  Procedure ClearConsoleHook;
  Property URL : String;
  Property BufferTimeout : Integer;
  Property HookConsole : Boolean;
end;
```

The following methods exist:

**Capture** This is the central call: the string aLine is sent to the server. If NewLine is set to True, a newline character is added.

**Flush** if the BufferTimeout is set to a positive number, lines will be buffered till the indicated timeout is reached. Flush will empty the buffer and send the contents to the server.

**SetConsoleHook** When calling this, the console hook will be installed, which means that all Write(Ln) statements will be written to the debug capture output as well. if a previous console hook was present, it will also be called.

**ClearConsoleHook** Resets the console hook to the state previous to calling SetConsoleHook

**SetExceptionsHook** When calling this, the OnShowException hook in SysUtils will be installed, which means that all calls to ShowExceptions will write to the debug capture output as well. If a previous console hook was present, it will also be called.

**ClearExceptionsHook** Resets the exceptions hook to the state previous to calling SetExceptionsHook

In addition, the following properties exist:

**Instance** This class property provides a standard instance, which is ready for you to configure and use.

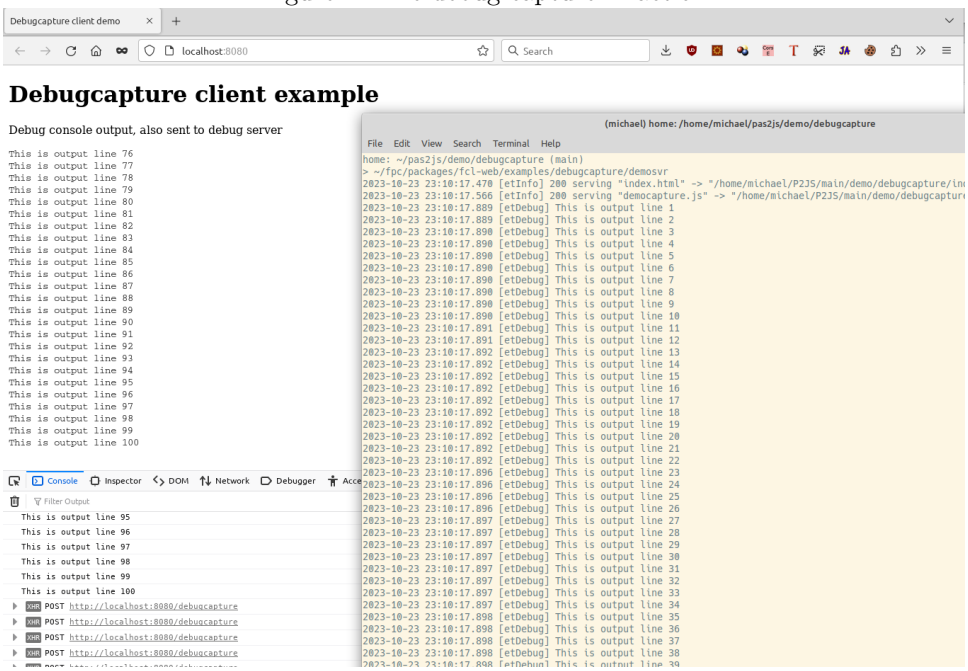
**URL** The URL to which all debug output is sent. The default URL is '/debug-capture'.

**BufferTimeout** A time (in milliseconds) during which log output is buffered locally before sending it to the server. If set to 0, then no buffering takes place, all logging is sent to the server immediatly.

**HookConsole** If set to True, then SetConsoleHook is called. If set to False, ClearConsoleHook is called.

**HookExceptins** If set to True, then SetExceptionsHook is called. If set to False, ClearExceptionsHook is called.

Figure 1: The debug capture in action




The use of this component is again quite straightforward, as shown by the following example program:

```
program democapture;

{$mode objfpc}
{$h+}

uses
  sysutils, classes, browserconsole, debugcapture;

Var
  I : integer;

begin
  With TDebugCaptureClient.Instance do
    begin
    BufferTimeout:=100;
    HookConsole:=True;
    end;
  For I:=1 to 100 do
    Writeln('This is output line '+IntToStr(I))
end.
```

The result of the 2 programs combined is shown in figure 1 on page 6. In the background, the browser is visible with the output of the `WriteLn` statements as HTML and in the browser debug console. In the foreground, the console on which the HTTP server program was started is visible. It shows the URLs that were loaded, and the debug capture output.

# 5 Conclusion

Free Pascal and PasJS come equipped with simple tools to enable you to to collect debugging information from applications in production. As shown here, the code to achieve this is really simple, and the classes used in the process are easily extended with extra functionalities: you can add a threaded mechanism on the server for improved performance, you can store the logs in a database, send them to logstash, all with a single mechanism which also works out of the box without the need for extra code.