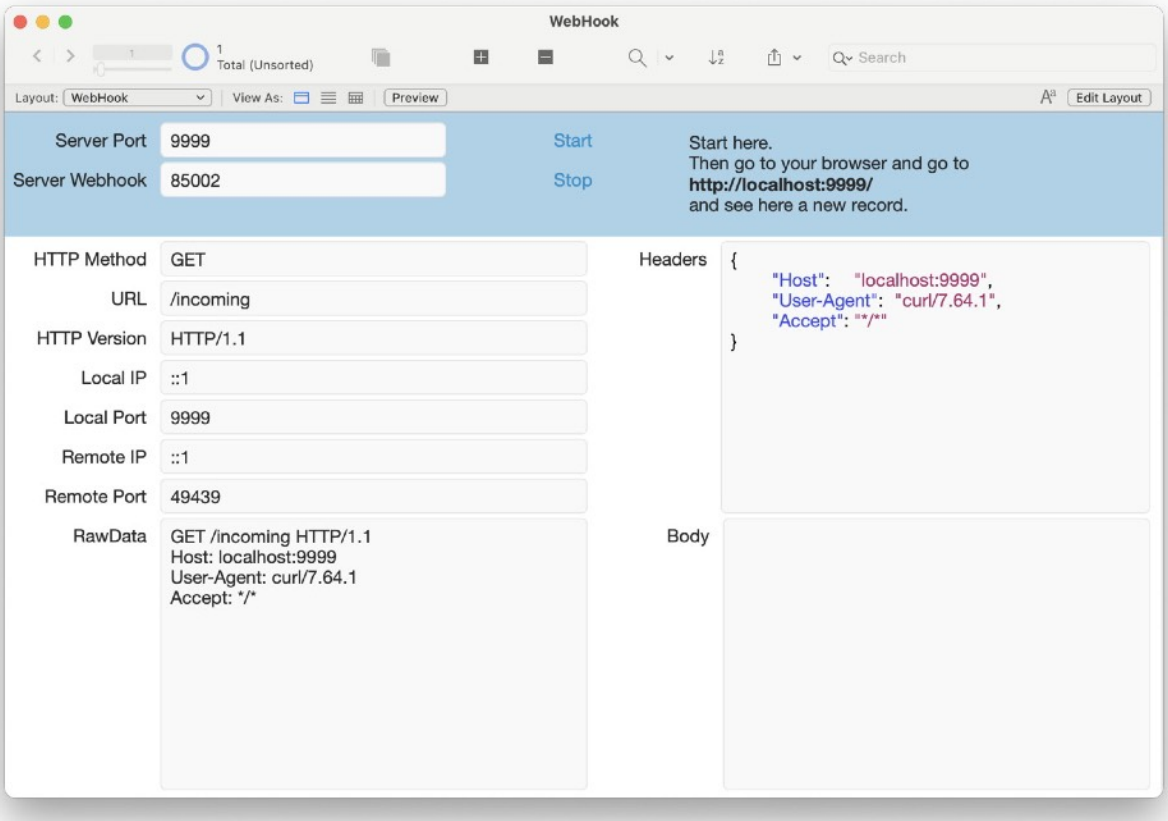
WebHook Introduction

Nowadays we live in a connected world, so web services run on different computers and sometimes need to notify each other. They usually do that with sending a HTTP request to a given URL and pass on some data as part of the URL or e.g. a JSON block as payload. FileMaker has no built-in web hook feature and with FileMaker Server you can do some things with DATA API or a custom php script on the web server part.

Our MBS FileMaker Plugin in version 11.5 comes with a new WebHook functionality. You can open a receiver to listen on a port for an incoming request and trigger a script to process the query. Our plugin provides maximum flexibility as you can create listeners as needed by script and stop them when the job is done.

Let's say you have a phone system, which can trigger you for a new incoming call. You configure for each phone an URL to trigger for that computer and have FileMaker listen on that computer for the given port. In our examples we use port 9999, but that could be any number chosen by you between 1024 and 65535. Please don't use ports other application want to use like port 5003 for FileMaker. If the IP on the local network for our workstation is 192.168.0.123, we could have the following URL triggered by the phone system:
http://192.168.0.123:9999/incomingcall

As you see we built an URL with http for the protocol. If needed, we could do https with TLS v1.2, but that needs a SSL certificate of course. If you like to simulate this, you could open a browser and enter the URL or run curl in the Terminal window:
curl -v http://localhost:9999/incomingcall



Setup Script

Let's get started to receive something in FileMaker. Make sure you have version 11.5 of our plugin installed, currently in beta testing. Create a new script to initialize the web hook like this:

Set Variable [$$WebHook ; Value: MBS("WebHook.Create")]
Set Variable [$r ; Value: MBS("WebHook.SetScript"; $$WebHook; Get(FileName); "WebHookReceived")]
Set Variable [$r ; Value: MBS("WebHook.Listen"; $$WebHook; 9999)]

You create a web hook object with our plugin using the WebHook.Create function. Then you can configure it. We just set the script to trigger, but you could also set certificate and key for TLS encryption. Once everything is setup, we call WebHook.Listen to start the underlaying socket on port 9999. If that port is not available, we get an error and may check if we can use another port or close the previous instance still listening on that port. A port can only be used by one application or service at a time.

Script Trigger

Next we create a script called "WebHookReceived" to be triggered. This script should take the reference number for the request and then we use it to query the URL we received:

```
Set Variable [ $WebRequest ; Value: Get(ScriptParameter) ]
Set Variable [ $URL ; Value: MBS("WebRequest.GetURL"; $WebRequest) ]
Set Variable [ $Body ; Value: MBS("WebRequest.GetBody"; $WebRequest; "UTF-8") ]
Set Variable [ $Headers ; Value: MBS("WebRequest.GetHeaders"; $WebRequest) ]
Set Variable [ $r ; Value: MBS("WebRequest.Release"; $WebRequest) ]
```

Based on the incoming URL or the payload, we can decide what to do, e.g. show a card window on the current layout.

Custom response

The example above is based on the plugin accepting the call and responding directly with an automatic answer, see WebHook.SetAutoAnswer. If you don't like to answer all requests with the same automatically answer, you can send a response yourself. For that we can allow you to leave the connection open when we trigger the script and you answer yourself. For that we disable auto answer and set the mode to keep connection open:

```
# enable keep open and disable auto answer, so we can send custom answer
Set Variable [ $r ; Value: MBS("WebHook.SetAutoAnswer"; $$Webhook; "") ]
Set Variable [ $r ; Value: MBS("WebHook.SetMode"; $$WebHook; 1) ]
```

Once that happened, we can change our script to send a custom answer to the other side. In this case we send a HTTP response with status code 201, our server name and finally a double line ending to mark the end. Be aware to send this with CRLF as line ending, so we use Text.ReplaceNewline here:

```
# send HTTP Response here
Set Variable [ $text ; Value: "HTTP/1.1 201 Created¶Server: MyServer 1.0¶Connection: close¶Content-Length: 0¶¶" ]
Set Variable [ $text ; Value: MBS( "Text.ReplaceNewline"; $Text; 3 ) ]
Set Variable [ $r ; Value: MBS("WebRequest.Send"; $WebRequest;
```

```
$text; "UTF-8") ]
# we do a little delay to make sure response is sent.
Pause/Resume Script [ Duration (seconds): ,1 ]
```

Be aware that the script handling this must run within a few seconds after the request comes in, otherwise the server on the other side may not wait for your response. For a lot of systems the auto responder is a convenient way to not run into timeout issues. The plugin acknowledges that the request was received and you can process the request later when there is time.

On FileMaker Server

If you run this on FileMaker Server, please be aware that script trigger won't work. You need to run a scheduled script with calls to WebHook.Check to process calls. This may be a script running every 10 minutes and then exists after 9:55 minutes. Or maybe this is part of a bigger script, which first sends a request to a web services then spins up the web hook to have it receive the answer callback later.

The following example script checks for status of incoming requests 10 times per second to trigger the receiver script for each ID we get:

```
Loop
    # check for pending requests
    Set Variable [ $list ; Value: MBS( "WebHook.Check" ) ]
    #
    # loop over pending requests
    Set Variable [ $count ; Value: ValueCount($list) ]
    Set Variable [ $index ; Value: 1 ]
    If [ $index ≤ $count ]
        Loop
            Set Variable [ $ref ; Value: GetValue($list; $index) ]
            Perform Script [ Specified: From list ; “WebHookReceived on
server” ; Parameter: $ref ]
            #
            # next
            Set Variable [ $index ; Value: $index + 1 ]
            Exit Loop If [ $index > $count ]
        End Loop
    End If
    #
    Pause/Resume Script [ Duration (seconds): ,1 ]
End Loop
```

If you need more, check out the other options we have including RAW mode, which can help if the data is not a HTTP request, but maybe some other device with it's own protocol, like a scale to measure your weight.
These WebHook functions can also help to trigger a script in FileMaker Pro from a script running on a server, take a call from Claris Connect or take a call from our own PHP scripts on another computer.
Let us know if you have questions.