# Webhandler Coding Cheatsheet

This is a FAQ / cheat-sheet for coding the webhandler

## Reading information from the HTTP Request

The request that is passed into a WebHandler is an implementation of the OpenEdge.Web.IWebRequest interface. API documentation for this interface is at <https://documentation.progress.com/output/oehttpclient/117/OpenEdge.Web.IWebRequest.html>

## Query strings

If you know the name of a query string, you can get the value by calling the GetQueryValue method on the request's URI property. The URI contains information about the URI used to make the request, including the schema (http), the host, port, path and query string.

Query strings are READ-ONLY.

|  |
| --- |
| method override protected integer HandleGet(input pReq as OpenEdge.Web.IWebRequest ):  define variable qryVal as character no-undo.  define variable names as character extent no-undo.    // Given request of `GET http://localhost:8810/api/web/talks?filter=id=ABL-010&top=2  pReq:URI:GetQueryNames(output names).  // Returns ["filter", "top"]  assign qryVal = pReq:URI:GetQueryValue('filter').  // returns "id=ABL-010"  assign qryVal = pReq:URI:GetQueryValue('skip').  // returns ? |

## Path Parameters

Path parameters are the {} enclosed values on the handler definition in the properties file, for example the mapping handler3 = Conference.SI.TalksHandler : /talks/{talk-id}/{stream-id}

Contains two path parameters: talk-id and stream-id

We can inspect the values, and enumerate the path parameters from the request.

Path parameters are READ-ONLY.

|  |
| --- |
| method override protected integer HandleGet(input pReq as OpenEdge.Web.IWebRequest ):  message pReq:PathParameterNames.  // Shows talk-id,stream-id    message pReq:UriTemplate.  // Shows /talks/{talk-id}/{stream-id}    // Given request of `GET http://localhost:8810/api/web/talks/ABL-010  message pReq:GetPathParameter('talk-id').  // Shows ABL-010 |

## Header values

HTTP headers are request (and response) metadata. They have a name, a value and possible parameters (with a delimiter, often ;).

Headers (names and values) are READ-WRITE.

Reading a request header

|  |
| --- |
| method override protected integer HandleGet(input pReq as OpenEdge.Web.IWebRequest ):  define variable hdrContentType as HttpHeader no-undo.    // get the header if it exists  if pReq:HasHeader('Content-Type':u) then  do:  assign hdrContentType= pReq:GetHeader('Content-Type':u).  // Display the entire value  message hdrContentType:Value.  // Shows text/plain; encoding=UTF-8  message hdrContentType:GetParameterValue('encoding').  // Shows UTF-8  end. |

Writing a response header

|  |
| --- |
| define variable resp as WebResponse no-undo.  assign resp = new OpenEdge.Web.WebResponse().  //EXAMPLE SETTING THE Location HEADER TO something like api/web/talks/ABL-018  resp:SetHeader('Location', 'api/web/talks/' + talkId). |

## Message body / entity

Many requests contain a message body (or entity). These entities contain a variety of data; the type of data is send as the request's Content-Type.

Message bodies are READ-WRITE.

Reading message bodies (JSON in this example)

|  |
| --- |
| method override protected integer HandleGet(input pReq as OpenEdge.Web.IWebRequest ):  define variable msgbody as Progress.Json.ObjectModel.JsonObject no-undo.  define variable streamId as character no-undo.    case pReq:ContentType:  when 'application/json' then  assign msgBody = cast(pReq:Entity, Progress.Json.ObjectModel.JsonObject).  otherwise  return error new Progress.Lang.AppError('Unable to convert message body', 0).  end case.    // now we can get values out of the body  if msgBody:Has('stream')  and msgBody:GetType('stream') eq Progress.Json.ObjectModel.JsonDataType:STRING  then  assign streamId = msgBody:GetCharacter('stream').    // Call business logic |

Writing message bodies

|  |
| --- |
| assign resp = new OpenEdge.Web.WebResponse()  msgBody = new Progress.Json.ObjectModel.JsonObject()  resp:Entity = msgBody  // we MUST set the ContentType property  resp:ContentType = 'application/json'    // a holder for the temp-table data  record = new Progress.Json.ObjectModel.JsonObject()  .  // we got ttTalk data from a call to business logic  buffer ttTalk:write-json('JsonObject', record, true).  msgBody:Add('data', record).  msgBody:Add('count', qryCnt).  // at some point we write the data out to the caller |