

HTTPAPI and NTLM

Adding and Using NTLM Authentication

Status of this document

Date: 16.07.2013
Version: 1.9

Preface

The authentication schemes supported by HTTPAPI are BASIC and DIGEST. These are the most common authentication methods used for HTTP transactions. NTLM authentication, which is often used by Microsoft's IIS servers, is not supported by default.

This document describes how to add NTLM authentication to HTTPAPI.

Warning

The NTLM extension has been developed by Thomas Raddatz and is not supported by Scott Klement who is the owner of HTTPAPI. Therefore always make sure that the NTLM patch level matches your version of HTTPAPI.

Also notice that the NTLM patch level will often be behind the current HTTPAPI beta version.

Prerequisites

A C compiler is required to compile the MD4C4 module. If there is no C compiler available, module MD4C4 is replaced by the RPG module MD4R4, which slightly decreases performance.

The minimum OS release level is V5R3M0.

Adding NTLM

Make sure that the NTLM patch matches your version of HTTPAPI. Source member NTLM_H contains a note about the supported HTTPAPI version number:

```
*=====*
*  NTLM: Authentication Plugin - Public  *
*=====*
*  Author   :  Thomas Raddatz           *
*  Date      :  21.05.2012              *
*  E-mail    :  thomas.raddatz@tools400.de *
*  Homepage:  www.tools400.de           *
*                                           *
*  HTTPAPI   :  1.25beta2               *
*=====*
```

Do not miss to check the version number, because you are going to replace module HTTPAPIR4!

Before you proceed to upload the provided source members to your HTTPAPI library, you should save the existing source members:

Source File	Member
QCLSRC	INSTALL
QRPGLESRC	HTTPAPI_H
QRPGLESRC	HTTPAPIR4
QRPGLESRC	HTTPUTILR4
QRPGLESRC	HTTPXMLR4
QSRVSR	HTTPAPI1
QSRVSR	HTTPAPI2
QRPGLESRC	ENCRYPTR4
QRPGLESRC	MD4_H
QRPGLESRC	MD4C4
QRPGLESRC	MD4R4
QRPGLESRC	NTLM_C
QRPGLESRC	NTLM_H
QRPGLESRC	NTLM_P
QRPGLESRC	NTLMR4

Feel free to use batch file `upload_httpapi_ntlm.bat` to upload the source members to library LIBHTTP. Please notice that the example and RPGUnit test programs are also uploaded to the i5 when using the batch file.

After having uploaded the source members you can compile and run the new installer to update your HTTPAPI to NTLM:

```
CRTCLPGM PGM(QTEMP/INSTALL)
  SRCFILE(LIBHTTP/QCLSRC) SRCMBR(*PGM)

CALL PGM(QTEMP/INSTALL)
```

Then rebuild WSDL2RPG to complete the installation:

```
CRTBNDCL PGM(QTEMP/A_INSTALL)
  SRCFILE(WSDL2RPG/QWSDL2RPG) SRCMBR(*PGM)

CALL PGM(QTEMP/A_INSTALL)
```

Starting with WSDL2RPG v1.15.2 it is no longer required to recompile WSDL2RPG.

Verifying NTLM

Starting with NTLM v1.1.1 a debug message showing NTLM version information has been added to the HTTPAPI debug log. Sample debug message:

```
HTTPAPI Ver 1.25beta2 released 2012-03-06
NTLM Ver 1.1.1 released 2012-09-28
OS/400 Ver V6R1M0
```

See also chapter: [Sample Web Service](#)

Using NTLM

Basically there are no differences between using BASIC or DIGEST and NTLM authentication.

Once that your GET or POST operations ends with a 401 authentication error you should call `http_getAuth()` as usually. Please notice that there is no dedicated parameter for NTLM authentication for NTLM v1.1.1 and before. But if the procedure returns a '0' for success, you can assume that NTLM authentication is active in case 'peBasic' and 'peDigest' are set to '*OFF'. Feel free to determine the active authentication scheme like this:

```
rc = http_url_get(...);
if (rc <> 1);
    http_error(err);
    if (err = HTTP_NDAUTH);
        rc = http_getAuth(isBasic: isDigest: realm);
        select;
        when (isBasic);
            http_setAuth(HTTP_AUTH_BASIC
                          : getUser(): getPassword());
        when (isDigest);
            http_setAuth(HTTP_AUTH_MD5_DIGEST
                          : getUser(): getPassword());
        other;
            http_setAuth(HTTP_AUTH_NTLM
                          : getUser(): getPassword());
        ends1;
        rc = http_url_get(...);
    endif;
endif;
```

The drawback of the missing parameter is that the stub cannot use NTLM authentication in case BASIC or DIGEST authentication is also active. For the following example the stub chooses BASIC authentication although NTLM is more secure and also available:

```
POST /HelloWorld.asmx HTTP/1.1
Host: 10.115.14.91
User-Agent: http-api/1.24
Content-Length: 227
Content-Type: text/xml
SOAPAction: "http://tempuri.org/HelloWorld"
```

```
sendraw(): entered
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope ...></soapenv:Envelope>
recvresp(): entered
HTTP/1.1 401 Unauthorized
Content-Type: text/html
Server: Microsoft-IIS/7.5
WWW-Authenticate: Basic realm="Test"
WWW-Authenticate: Negotiate
WWW-Authenticate: NTLM
X-Powered-By: ASP.NET
```

Starting with v1.2 parameter 'peNTLM' can be used to test for NTLM authentication.

Realm

Also notice that NTLM actually does not use a realm at all. Instead of an empty realm, the NTLM authentication module returns the host name as the realm value.

The NTLM module always returns the host name as a substitute of the realm.

Domain

On the other hand NTLM knows something that is called a “domain”. A user is assigned to a domain just the way your email address belongs to a domain. Since BASIC and DIGEST authentication do not use a domain at all, HTTPAPI does not provide any option to specify a domain for authentication.

Usually there is no need to specify the domain because in most cases the server belongs to the same domain as the users who use the services of the server.

But in case you need to specify a domain you can prefix the user name with the domain name, separated by a backslash as shown here:

domain\username

Or you can use the following format:

username@domain

A problem you may be faced with might be, that HTTPAPI restricts the length of the user name to 80 characters, which might be too short to take a domain and a user name. You can get around that problem by specifying the user name and password in the URL as shown here:

http://domain\username:password@the.hostname.com/index.html

LAN Manager Compatibility Modes

The LAN Manager Compatibility Mode is controlled by variable 'g_LMCompatibility' of module NTLMR4.

The default mode is NTLMv2. Beside NTLM V2 the following modes are supported:

- 0 Sends NTLMv1 responses. That may also include the cryptographically-weak LM response.
(Flags: NTLMSSP_NEGOTIATE_NT_ONLY + NTLMSSP_NEGOTIATE_NTLM2)
- 1 Sends NTLMv1 responses but the cryptographically-weak LM response is excluded. The communication fails when the server does not support NT_ONLY or NTLM2.
(Flags: NTLMSSP_NEGOTIATE_NT_ONLY + NTLMSSP_NEGOTIATE_NTLM2)
- 2 Sends NTLMv1 responses but enforces NTLM2.
(Flags: NTLMSSP_NEGOTIATE_NTLM2)
- 3 Sends NTLMv2 responses. Recommended default mode.

NTLM Protocol

In contrast to BASIC and DIGEST authentication NTLM authenticates a connection and not a request. Since authenticating a user using NTLM is more expensive than BASIC or DIGEST authentication, it is recommended to use a persistent connection and no single GET or POST request in case it is required to multiple requests to the server.

HTTPAPI exposes a bunch of procedures all starting with `http_persist_*` to manage persistent connections.

BASIC/DIGEST Handshake

BASIC or DIGEST authentication requires a total of 4 messages between the client and server to get a file from the server.

```
GET /basic/index.html HTTP/1.1
```

Since the client is not yet authorized the server returns a 401 “Unauthorized” status code along with a “WWW-Authenticate” header to let the client know the supported authentication scheme:

```
HTTP/1.1 401 Unauthorized
```

```
WWW-Authenticate: Basic realm="Basic Authentication"
```

Now the client retrieves the user credentials and resends the same request enriched with an “Authorisation” header:

```
GET /basic/index.html HTTP/1.1
```

```
Authorization: Basic YWRtaW46bXlTZWNyZXQ=
```

Eventually the server verifies the user credentials and returns the requested resource along with a 200 “OK” status code:

```
HTTP/1.1 200 OK
```

NTLM Handshake

NTLM however requires a total of 6 messages to get the first resource from the server. Again the communication starts with an initial request for a resource:

```
POST /HelloWorld.aspx HTTP/1.1
```

The server returns a 401 “Unauthorized” status code along with a “WWW-Authenticate” header requesting NTLM authentication:

```
WWW-Authenticate: NTLM
```

Now the two additional messages come into play to negotiate the authentication details. First the client sends a NEGOTIATE message to the server:

```
GET /HelloWorld.aspx HTTP/1.1
```

```
Authorization: NTLM <base64-encoded type-1-message>
```

The server response to that with a CHALLENGE message:

```
HTTP/1.1 401 Unauthorized
```

```
Authorization: NTLM <base64-encoded type-2-message>
```

Eventually the client uses an AUTHENTICATE (Type-3) message to get the resource from the server:

```
POST /HelloWorld.aspx HTTP/1.1
```

```
Authorization: NTLM <base64-encoded type-3-message>
```

If everything is fine, the server returns the requested resource to the client:

```
HTTP/1.1 200 OK
```

From now on the client can continue requesting resources as long as the connection is established without going through the complete authentication process.

Please have in mind that connections are not persistent when using HTTPAPI's GET or POST procedures. The only way to establish a persistent connection with HTTPAPI is to use the `http_persist_*` procedures which are slightly more complex than `http_url_get()` or `http_url_get()`.

Debugging NTLM Authentication Headers

Debugging connection problems always starts with the HTTP API debug log. The HTTP API debug log contains the complete HTTP data flow that was exchanged between the client and the server. Sometimes you may need to know the content of the NTLM authentication headers, e.g. you may be asked about the NTLM negotiating flags that were used. These headers are Base64 encoded data that is not easy to analyse.

For that the “NTLM Message Inspector” comes into play. The NTLM Message Inspector lets you easily look at the content of a given NTLM authentication header. It is shipped as a self-executing jar file:

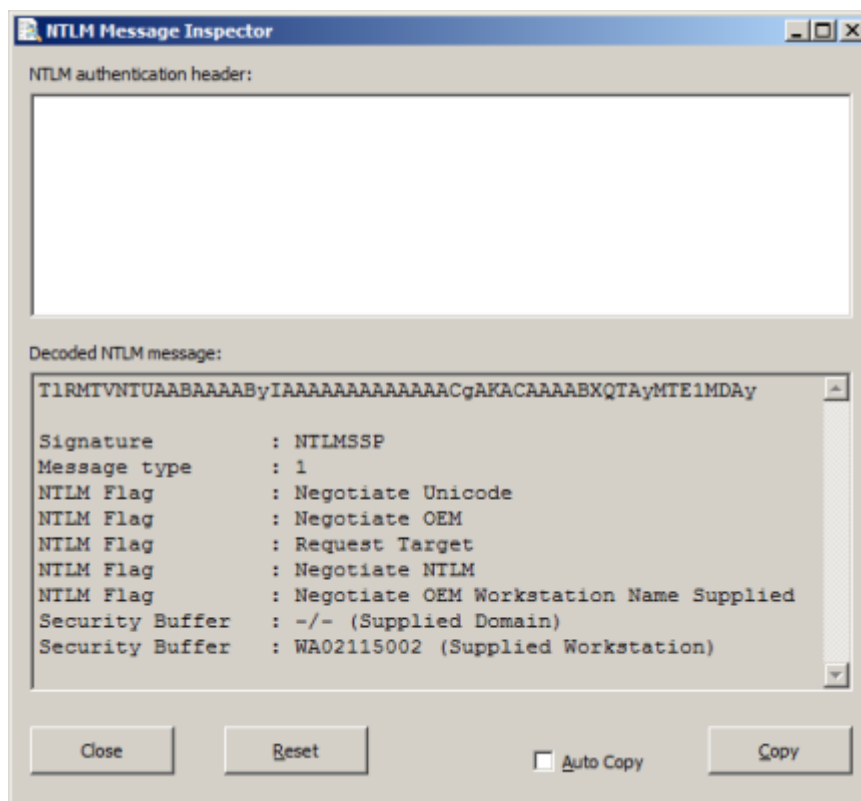
```
NTLMMessageInspector.jar
```

Double click the jar file to start the inspector. If that does not work, make sure you have a JRE 1.6 or higher installed and try to start it from the command line:

```
java -cp NTLMMessageInspector.jar  
de.tools400.net.ntlm.analyzer.NTLMMessageInspector
```

When the inspector has been started, copy the authentication header in question into the input field “NTLM authentication header”. You do not necessarily need to remove the “Authorization: NTLM” prefix from the header. Just paste the complete line as shown in the example below:

```
GET /ntlm.html HTTP/1.1  
Host: 10.115.14.91:80  
User-Agent: http-api/1.24  
Authorization: NTLM TlRMTVNTUAABAAAByIAAAAAAAAAAAAAACgAKACAAAABXQTAyMTE1MDAy
```



The input field is cleared and the result is shown below.

Example: Persistent Connection

This example demonstrates how to call a web service twice, storing the response SOAP messages in a single stream file. The complete source code is shipped with the NTLM package.

```
URL = 'http://' + Job_getTcpIpAddr() + '/HelloWorld.asmx';
IFS = '/home/raddatz/httpapi_example37.xml';

// Open output file
fd = open(IFS: O_WRONLY + O_TRUNC + O_CREAT + O_CSID: 511: 1208);

// Produce SOAP message
postData =
    '<soapenv:Envelope +
      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" +
      xmlns:tem="http://tempuri.org/">+
      <soapenv:Header/>+
      <soapenv:Body>+
        <tem:HelloWorld/>+
      </soapenv:Body>+
    </soapenv:Envelope>';

// Open persistent connection
pComm = http_persist_open(URL);

// Set credentials
http_setauth(HTTP_AUTH_NTLM: user: password);

do '1';

    // Call web service
    if ( http_persist_post( pComm: URL: 0: *null
                          : %addr(postData)+2: %len(postData)
                          : fd: %paddr('write')) = -1);

        leave;
    endif;

    // Call web service once more
    if ( http_persist_post( pComm: URL: 0: *null
                          : %addr(postData)+2: %len(postData)
                          : fd: %paddr('write')) = -1);

        leave;
    endif;

enddo;

// Close http connection
http_persist_close(pComm);

// Close output file
callp close(fd);
```

Sample “HelloWorld” Web Service:

You can easily set up a web service with NTLM authentication. On a Windows PC just install and start the IIS service and drop the following file into folder “wwwroot”:

HelloWorld.asmx

```
<%@ WebService Language="C#"
Class="ProgWS.Ch02.HelloWorldService" %>
using System.Web.Services;
namespace ProgWS.Ch02
{
    public class HelloWorldService: WebService
    {
        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }
    }
}
```

Refer to [Installing IIS on Windows 7](#) for a brief description of how to install IIS 7.5 on Windows 7.

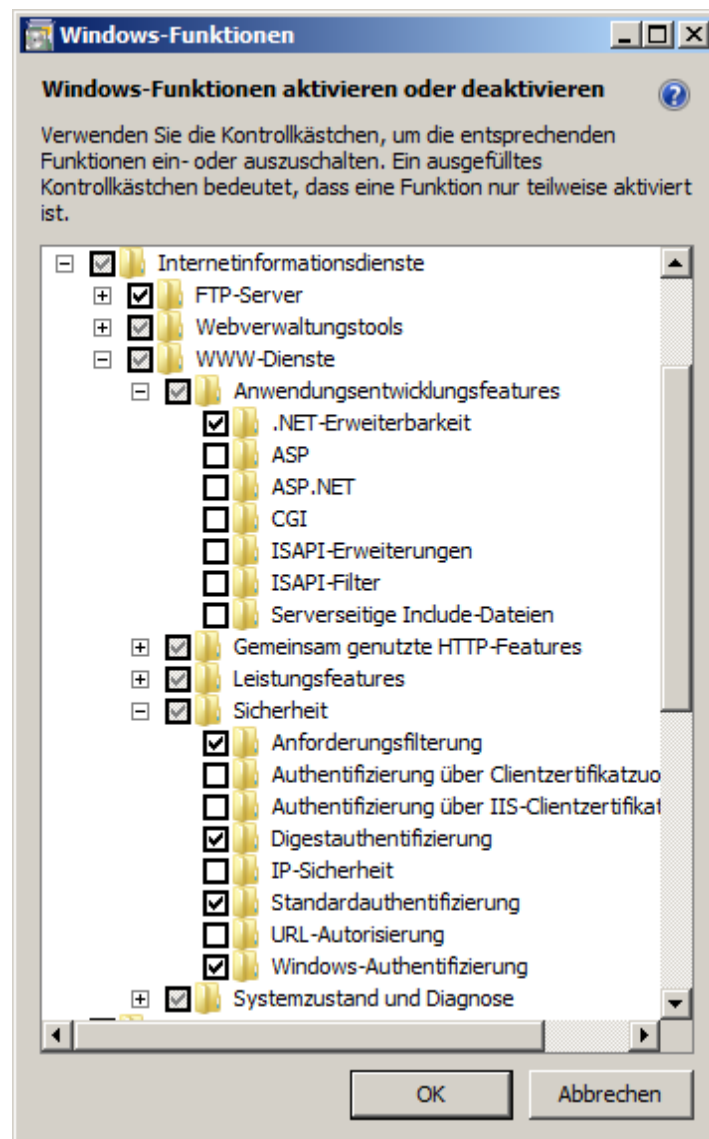
Installing IIS on Windows 7

The following description is part of the Microsoft TechNet Library at <http://technet.microsoft.com/en-us/library/cc731911.aspx>:

By default, IIS 7.5 is not installed on Windows® 7. You can install IIS by clicking **Windows Features** in **Advanced Options** under **Programs** in **Control Panel**.

You can perform this procedure using the user interface (UI) or a script.

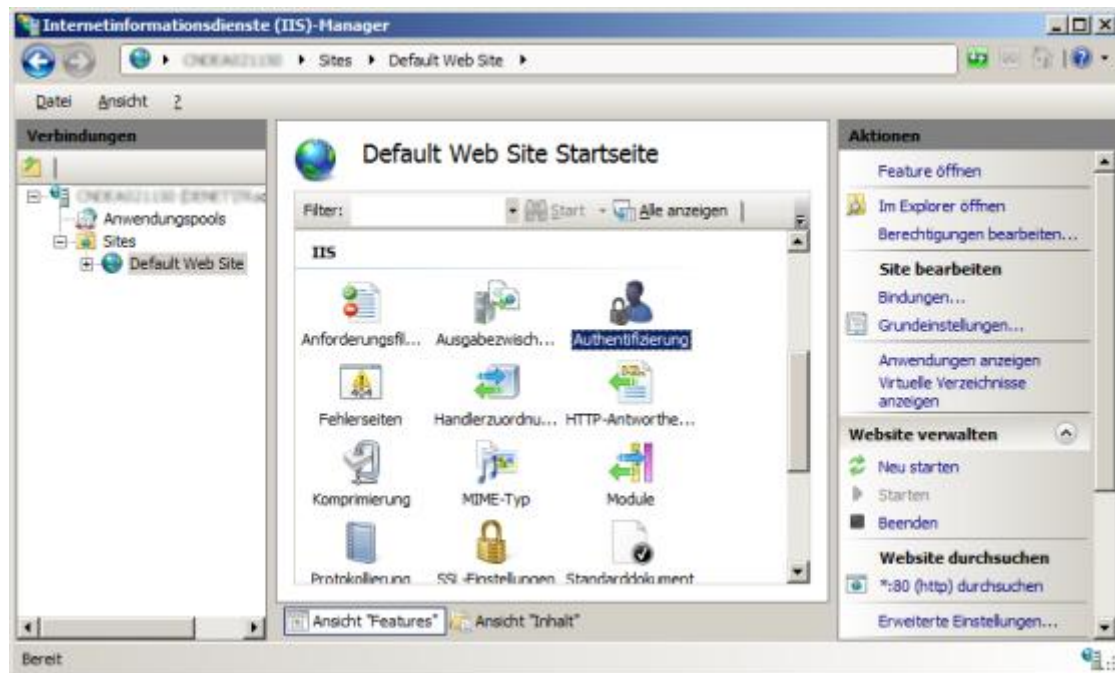
1. Click **Start** and then click **Control Panel**.
2. In **Control Panel**, click **Programs** and then click **Turn Windows features on or off**.
3. In the **Windows Features** dialog box, click **Internet Information Services** and then click **OK**.



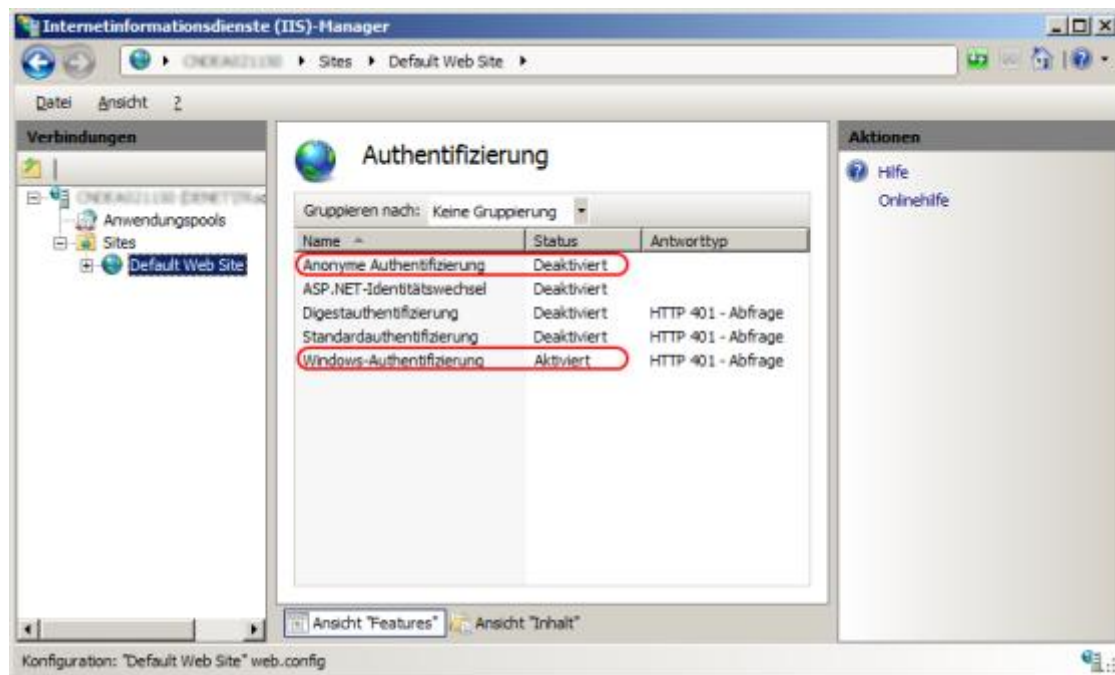
Sorry for the German screen shot. I do not have an English Windows 7.

Make sure to turn off „Anonymous Authentication“ and to enable “Windows Authentication.

1. Click **Start** and then click **Control Panel**.
2. In **Control Panel**, click **System and Security** and then click **Administrative Tools**.
3. In the **Administrative Tools** window, double-click **Internet Information Services (IIS) Manager**.



(<http://technet.microsoft.com/en-us/library/cc770472%28v=ws.10%29.aspx>)



(<http://technet.microsoft.com/en-us/library/cc754628%28v=ws.10%29.aspx>)

Sorry for the German screen shots. I do not have an English Windows 7.

NTLM Proxy Authentication

Currently NTLM proxy authentication is not supported by the NTLM extension of HTTPAPI. The reason is that I do not have a proxy server that uses NTLM authentication and therefore cannot test it.

Examples

- EXAMPLE25 This example demonstrates how to use `http_url_get()` to get a web page that is protected by NTLM authentication.
- EXAMPLE26 This example demonstrates how to use `http_persist_get()` to get a web page that is protected by NTLM authentication. The advantage of this example is that you can call `http_persist_get()` multiple times once that you passed the authentication process.
- EXAMPLE27 This example demonstrates how to use `http_persist_get()` to get a web page that is protected by NTLM authentication. This time `http_setauth()` is directly called after `http_persist_open()` to minimize the overhead of the authentication process.
- EXAMPLE35 This example demonstrates how to use `http_url_post()` to call a web service using NTLM authentication.
- EXAMPLE37 This example demonstrates how to use `http_persist_post()` to call a web service using NTLM authentication.
- EXAMPLE38 This example demonstrates how to use `http_persist_post()` to call a web service using NTLM authentication and parse the response with eXpat. **Experimental!**

Your comments are important to me! Any comments sent to me are greatly appreciated.

thomas.raddatz@tools400.de