Novell Developer Kit

www.novell.com

NETWARE® REMOTE MANAGER SDK

October 8, 2003

Novell®

Legal Notices

Novell, Inc.
1800 South Novell Place
Provo, UT 84606
U.S.A.

www.novell.com

NDK NetWare Remote Manager SDK
October 8, 2003

Online Documentation: To access the online documentation for this and other Novell developer products, and to get updates, see developer.novell.com/ndk. To access online documentation for Novell products, see www.novell.com/documentation.

Novell Trademarks

AlarmPro is a registered trademark of Novell, Inc. in the United States and other countries.
AppNotes is a registered trademark of Novell, Inc.
AppTester is a registered trademark of Novell, Inc. in the United States.
ASM is a trademark of Novell, Inc.
BorderManager is a registered trademark of Novell, Inc.
BrainShare is a registered service mark of Novell, Inc. in the United States and other countries.
C3PO is a trademark of Novell, Inc.
Client32 is a trademark of Novell, Inc.
ConsoleOne is a registered trademark of Novell, Inc.
Controlled Access Printer is a trademark of Novell, Inc.
Custom 3rd-Party Object is a trademark of Novell, Inc.
DeveloperNet is a registered trademark of Novell, Inc. in the United States and other countries.
digitalme is a registered trademark of Novell, Inc.
DirXML is a registered trademark of Novell, Inc.
eDirectory is a trademark of Novell, Inc.
exteNd is a trademark of Novell, Inc.
exteNd Composer is a trademark of Novell, Inc.
exteNd Director is a trademark of Novell, Inc.
exteNd Workbench is a trademark of Novell, Inc.
FAN-OUT FAILOVER is a trademark of Novell, Inc.
Full Service Directory is a trademark of Novell, Inc.
GroupWise is a registered trademark of Novell, Inc. in the United States and other countries.
Hardware Specific Module is a trademark of Novell, Inc.
Hot Fix is a trademark of Novell, Inc.
iChain is a registered trademark of Novell, Inc.
instantme is a registered trademark of Novell, Inc.
Internetwork Packet Exchange is a trademark of Novell, Inc.
IPX is a trademark of Novell, Inc.
IPX/SPX is a trademark of Novell, Inc.
jBroker is a trademark of Novell, Inc.
JNOS is a trademark of Novell, Inc.
Link Support Layer is a trademark of Novell, Inc.
LSL is a trademark of Novell, Inc.
ManageWise is a registered trademark of Novell, Inc., in the United States and other countries.
Media Support Module is a trademark of Novell, Inc.
Mirrored Server Link is a trademark of Novell, Inc.
MP/M is a trademark of Novell, Inc.
MSL is a trademark of Novell, Inc.
MSM is a trademark of Novell, Inc.
NCP is a trademark of Novell, Inc.
NDPS is a registered trademark of Novell, Inc.
NDS is a registered trademark of Novell, Inc. in the United States and other countries.
NDS Manager is a trademark of Novell, Inc.
NE2000 is a trademark of Novell, Inc.
NetMail is a trademark of Novell, Inc.
NetWare is a registered trademark of Novell, Inc. in the United States and other countries.
NetWare/IP is a trademark of Novell, Inc.
NetWare Core Protocol is a trademark of Novell, Inc.
NetWare Loadable Module is a trademark of Novell, Inc.
NetWare Management Portal is a trademark of Novell, Inc.
NetWare MHS is a registered trademark of Novell, Inc.
NetWare Name Service is a trademark of Novell, Inc.
NetWare Peripheral Architecture is a trademark of Novell, Inc.
NetWare Print Server is a trademark of Novell, Inc.

NetWare Requester is a trademark of Novell, Inc.
NetWare SFT and NetWare SFT III are trademarks of Novell, Inc.
NetWare SQL is a trademark of Novell, Inc.
NetWire is a registered service mark of Novell, Inc. in the United States and other countries.
NLM is a trademark of Novell, Inc.
NMAS is a trademark of Novell, Inc.
NMS is a trademark of Novell, Inc.
Novell is a registered trademark of Novell, Inc. in the United States and other countries.
Novell Application Launcher is a trademark of Novell, Inc.
Novell Authorized Service Center is a service mark of Novell, Inc.
Novell Certificate Server is a trademark of Novell, Inc.
Novell Client is a trademark of Novell, Inc.
Novell Cluster Services is a trademark of Novell, Inc.
Novell Directory Services is a registered trademark of Novell, Inc.
Novell Distributed Print Services is a trademark of Novell, Inc.
Novell iFolder is a registered trademark of Novell, Inc.
Novell Labs is a trademark of Novell, Inc.
Novell Replication Services is a trademark of Novell, Inc.
Novell SecretStore is a registered trademark of Novell, Inc.
Novell Security Attributes is a trademark of Novell, Inc.
Novell Storage Services is a trademark of Novell, Inc.
Novell Web Server is a trademark of Novell, Inc.
Novell, Yes, Tested & Approved logo is a trademark of Novell, Inc.
NSI is a trademark of Novell, Inc.
Nsure is a trademark of Novell, Inc.
Nterprise is a trademark of Novell, Inc.
Nterprise Branch Office is a trademark of Novell, Inc.
ODI is a trademark of Novell, Inc.
Open Data-Link Interface is a trademark of Novell, Inc.
Open Socket is a registered trademark of Novell, Inc.
Packet Burst is a trademark of Novell, Inc.
POSE is a trademark of Novell, Inc.
Printer Agent is a trademark of Novell, Inc.
QuickFinder is a trademark of Novell, Inc.
Red Box is a trademark of Novell, Inc.
Remote Console is a trademark of Novell, Inc.
RX-Net is a trademark of Novell, Inc.
Sequenced Packet Exchange is a trademark of Novell, Inc.
SFT and SFT III are trademarks of Novell, Inc.
SPX is a trademark of Novell, Inc.
Storage Management Services is a trademark of Novell, Inc.
System V is a trademark of Novell, Inc.
Topology Specific Module is a trademark of Novell, Inc.
Transaction Tracking System is a trademark of Novell, Inc.
TSM is a trademark of Novell, Inc.
TTS is a trademark of Novell, Inc.
Universal Component System is a registered trademark of Novell, Inc.
Virtual Loadable Module is a trademark of Novell, Inc.
VLM is a trademark of Novell, Inc.
ZENworks is a registered trademark of Novell, Inc.

Third-Party Trademarks

All third-party trademarks are the property of their respective owners.

Contents

NetWare Remote Manager SDK

The NetWare Remote Manager SDK provides a set of functions that allow your application to interact with a remote client using HTTP protocol. This interface can be used to extend the NetWare Remote Manager to include custom management and diagnostic information. Starting with NetWare 6.5, the interface also allows you to send HTTP client requests.

1 Tasks

This section contains common tasks and examples of the NetWare Remote Manager API. "Registering Your Service" on page 11 walks you through adding a service to the NetWare Remote Manager and registering this service with HTTP stack.

"Handling Requests" on page 12 shows you how to create a simple method to handle client requests and pass information back to the client. Once your service is registered and you can handle requests, you will be able to pass information to and from the NetWare Remote Manager.

Registering Your Service

Before you can handle HTTP requests, you need to register your service with HTTP stack. This allows HTTP stack to forward URL requests to the proper method. You can also list your service in the main table of contents for simplified access. Take a look at the NetWare Remote Manager entry page and decide if your service fits under one of the topics listed on the left.

There are two methods used to register services with HTTP stack; RegisterServiceMethod (page 24), and RegisterServiceMethodEx (page 26). RegisterServiceMethodEx contains a parameter to pass a table of contents structure.

Before you call RegisterServiceMethodEx, you need to build the TOCregistration structure and fill in the variables:

```
struct TOCregistration tocreg;
CStrCpy(&tocreg.TOCHeadingName, "Volume Management");
tocreg.TOCHeadingNumber = 0;
CSetB(0, &tocreg.reserved[0], 4);
```

TOCHeadingNumber corresponds with the seven headings listed on the left side of the entry page (starting with 0). You can list your service under any heading by setting TOCHeadingNumber to the corresponding value.

Now that you have the TOCregistration struct, you can register your service with the HTTP stack:

```
BOOL eflg;
char *pzDemoService = "DEMO";
eflg = RegisterServiceMethodEx("Portal Demo Pages", pzDemoService,
CStrLen(pzDemoService), &tocreg, 0, 0, &demoMethod, demoServiceMethodTag,
&ccode);
if (eflg == FALSE) return(ccode);
return(0);
```

You now have a service registered under the name DEMO and demoMethod is registered as the connection request method. Now, if a URL request comes in to http://<portal_server>/DEMO/..., demoMethod is called to handle the request. Note that you need to use RegisterServiceMethodEx in order to register your service in the table of contents for NetWare Remote Manager.

The first call to the registered method is an initialization call. The CONTROL_INITIALIZATION_BIT is set in the InformationBits to signal this call. This allows the method to do any of its own initialization it needs to do before handling HTTP requests. Also, when HTTPSTK unloads or the method is deregistered, another call to the method is made with the CONTROL_DEINITIALIZATION_BIT set. Make sure any symbols dynamically imported from HTTPSTK are released before returning from your method on deinitialization.

Handling Requests

When you registered your service method in the previous section ("Registering Your Service" on page 11), you specified demoMethod as your connection request method. This is the method that is called when a request comes in for your service.

In order to handle these requests, demoMethod needs to do more than just return a response; demoMethod needs to perform the following:

1. Check initialization and deinitialization.
2. Get the request (and ensure that it is valid).
3. Check the type of the request (and ensure that the request type is one handled by our service).
4. Get the path (and ensure that it is valid).
5. Process the path and return a response.

The following demoMethod is modeled after the pServiceMethod callback defined with RegisterServiceMethodEx (page 26). Parameter descriptions are included with RegisterServiceMethodEx:

```
UINT32 demoMethod(HINTERNET hndl, void *info, UINT32 infoLen, UNIT32
InfoBits)
```

The following code illustrates how to perform each of the tasks outlined above. You can then combine these snippets together and use it as a model for your own connection request method.

Variables

```
int err = 0;  //stores returned error value
int len = 0;  //determine if request is valid length
char *bp;     //stores path portion of URL
char* cnv;    //un-escaped path buffer
UINT32 urlcount = 0;    //
UINT32 methodType = 0;   //stores request type
```

Initialization

```
//check to see if the process needs to be initialized
//or deinitialized

if (InfoBits & CONTROL_INITIALIZATION_BIT) return(0);
if(InfoBits & CONTROL_DEINITIALIZATION_BIT) return(0);
```

Get the Request

```
//get the numerical value of the request method

if (HttpReturnRequestMethod(hndl, &methodType) != TRUE)
{
```

```
//hndl is invalid, return error
HttpSendErrorResponse(hndl, HTTP_INTERNAL_SERVER_ERROR);
return(HTTP_INTERNAL_SERVER_ERROR);
}
```

Check the Request Type

```
//determine if the request is a get
//we will only handle get requests

if (methodType != HTTP_REQUEST_METHOD_GET)
{
//request is not a get, return error
HttpSendErrorResponse(hndl, HTTP_STATUS_NOT_SUPPORTED);
return(HTTP_STATUS_NOT_SUPPORTED);
}
```

Get the Requested Path

```
//get the requested path, remove our service name
//and make sure that the request is valid

err = HttpReturnPathBuffers(hndl, NULL, &bp, &cnv);
if (err)
{
//hndl is invalid, return error
HttpSendErrorResponse(hndl, HTTP_INTERNAL_SERVER_ERROR);
return(HTTP_INTERNAL_SERVER_ERROR);
}
```

Process the Request and Return a Response

```
//convert the component path to a usable form

if (*bp != 0)
{
   //bp++;
   urlCount = ConvertToComponentPath(bp);
}

//switch statement to determine the response
//will display demo pages 2, 3, 4 or entry

err = 0;
switch (urlCount)
{
case 1:// Display demo pages 2, 3, or 4
if(ICmpB(bp, pzDemoPage2, (*bp) + 1) == -1)
{
HttpSendSuccessfulResponse(hndl, HttpReturnString(HTTP_CONTENT_TYPE_HTML));

   HttpSendDataSprintf(hndl,"<HTML><HEAD><TITLE>DEMO Page 2</TITLE>
   %s</HEAD><BODY>", PORTAL_STYLE_SHEET);
   HttpSendDataSprintf(hndl, "<H2>This is demo page 2.</BODY>",
   PORTAL_STYLE_SHEET);
   break;
}
if(ICmpB(bp, pzDemoPage3, (*bp) + 1) == -1)
{
   HttpSendSuccessfulResponse(hndl,
   HttpReturnString(HTTP_CONTENT_TYPE_HTML));
```

```
      HttpSendDataSprintf(hndl, "<HTML><HEAD><TITLE>DEMO Page 3
      <TITLE>%s</HEAD><BODY>", PORTAL_STYLE_SHEET);
      HttpSendDataSprintf(hndl, "<H2>This is demo page 3.</BODY>",
      PORTAL_STYLE_SHEET);
      break;
   }
   if(ICmpB(bp, pzDemoPage4, (*bp) + 1) == -1)
   {
      HttpSendSuccessfulResponse(hndl,
   HttpReturnString(HTTP_CONTENT_TYPE_HTML));
      HttpSendDataSprintf(hndl, "<HTML><HEAD><TITLE>DEMO Page 4
      <TITLE>%s</HEAD><BODY>", PORTAL_STYLE_SHEET);
      HttpSendDataSprintf(hndl, "<H2>This is demo page 4.
      </BODY>", PORTAL_STYLE_SHEET);
      break;
   }

   err = HTTP_STATUS_BAD_REQUEST;
   break;

      case 0:// display server page
      HttpSendSuccessfulResponse(hndl,
      HttpReturnString(HTTP_CONTENT_TYPE_HTML));
      HttpSendDataSprintf(hndl, "<HTML><HEAD><TITLE>DEMO Page 1
      </TITLE>%s</HEAD><BODY>", PORTAL_STYLE_SHEET);
      HttpSendDataSprintf(hndl, "<H1>This is demo page 1.</H1>",
      PORTAL_STYLE_SHEET);

      HttpSendDataSprintf(hndl, "<B><A HREF=/%s/%s>Demo Page 2
      </A>\n", pzDemoService, pzDemoPage2+1);
      HttpSendDataSprintf(hndl, "<B><A HREF=/%s/%s>Demo Page
      </A>\n", pzDemoService, pzDemoPage3+1);
      HttpSendDataSprintf(hndl, "<B><A HREF=/%s/%s>Demo Page 4
      </A>\n", pzDemoService, pzDemoPage4+1);
      HttpSendDataSprintf(hndl, "</BODY>");
      break;

      default:// URL passed was bad - bail out
      err = HTTP_STATUS_BAD_REQUEST;
      break;
   }
   if(err)
      HttpSendErrorResponse(hndl, err);
   else
      HttpEndDataResponse(hndl);

   return(err);
   }
```

demoMethod will now return simple HTML reply to requests to our service.

Demo Method

The following is a complete version of the demoMethod sample, including methods to convert the URL path.

```
//include statements

#include "httpexp.h"
```

```
char *pzDemoService = "DEMO";  //the registered service name
char *pzDemoPage2 = "\x3pg2";
char *pzDemoPage3 = "\x3pg3";
char *pzDemoPage4 = "\x3pg4";

//utility functions to parse the URL

#define ConvertSlashToNull(bp, cnt) {cnt = 0; while (bp[cnt])
{if (bp[cnt] == '/') {bp[cnt] = 0;}cnt++;}}
#defineCountToSlash(bp, cnt) {cnt = 0; while (*bp && (*bp == '/')){cnt++;}}

//HTML tag to use Portal style sheets

char * PORTAL_STYLE_SHEET = "<LINK REL=stylesheet TYPE=""text/css"" HREF=/SYS/LOGIN/
portal.css>";

//method to process the requested path

int ConvertToComponentPath(char *dst)
{
 int cnt;
 char *pcnt;
 char c;

 for (cnt=0, c=*dst; ((c != 0) && (c != '?') && (c !='=') && (c !='#'));)
  {
    pcnt = dst++;
    *pcnt = 0;
    c=*dst;
    if (c)
     {
       for (; ((c != 0) && (c != '/') && (c != '?') && (c !='=') && (c !='#'));)
        {
          dst++;
          c = *dst;
          (*pcnt)++;
        }
cnt++;
     }
  }
return(cnt);
}



//process the get request using our connection request method, demo method.

int err = 0;  //stores returned error value
int len = 0;  //determine if request is valid length
char *bp;     //stores path portion of URL
char* cnv;    //un-escaped path buffer

UINT32 urlcount = 0;     //
UINT32 methodType = 0;   //stores request type

//check to see if the process needs to be initialized
//or deinitialized

if (InfoBits & CONTROL_INITIALIZATION_BIT) return(0);
if (InfoBits & CONTROL_DEINITIALIZATION_BIT) return(0);
```

```
//get the numerical value of the request method

if (HttpReturnRequestMethod(hndl, &methodType) != TRUE)
{
   //hndl is invalid, return error
   HttpSendErrorResponse(hndl, HTTP_INTERNAL_SERVER_ERROR);
   return(HTTP_INTERNAL_SERVER_ERROR);
}
//determine if the request is a get
//we will only handle get requests

if (methodType != HTTP_REQUEST_METHOD_GET)
{
   //request is not a get, return error
   HttpSendErrorResponse(hndl, HTTP_STATUS_NOT_SUPPORTED);
   return(HTTP_STATUS_NOT_SUPPORTED);
}

//get the requested path, remove our service name
//and make sure that the request is valid

err = HttpReturnPathBuffers(hndl, NULL, &bp, &cnv);
if (err)
{
//hndl is invalid, return error

   HttpSendErrorResponse(hndl, HTTP_INTERNAL_SERVER_ERROR);
   Return(HTTP_INTERNAL_SERVER_ERROR);
}

//convert the component path to a usable form

if (*bp != 0)
{
   //bp++;
   urlCount = ConvertToComponentPath(bp);
}

//switch statement to determine the response
//will display demo pages 2, 3, 4 or entry

err = 0;
switch (urlCount)
{
case 1:// Display demo pages 2, 3, or 4
if(ICmpB(bp, pzDemoPage2, (*bp) + 1) == -1)
{
HttpSendSuccessfulResponse(hndl, HttpReturnString(HTTP_CONTENT_TYPE_HTML));

   HttpSendDataSprintf(hndl,"<HTML><HEAD><TITLE>DEMO Page 2</TITLE>
   %s</HEAD><BODY>", PORTAL_STYLE_SHEET);
   HttpSendDataSprintf(hndl, "<H2>This is demo page 2.</BODY>",
   PORTAL_STYLE_SHEET);
   break;
}
if(ICmpB(bp, pzDemoPage3, (*bp) + 1) == -1)
{
   HttpSendSuccessfulResponse(hndl,
   HttpReturnString(HTTP_CONTENT_TYPE_HTML));
   HttpSendDataSprintf(hndl, "<HTML><HEAD><TITLE>DEMO Page 3
```

```
   <TITLE>%s</HEAD><BODY>", PORTAL_STYLE_SHEET);
   HttpSendDataSprintf(hndl, "<H2>This is demo page 3.</BODY>",
   PORTAL_STYLE_SHEET);
   break;
}
if(ICmpB(bp, pzDemoPage4, (*bp) + 1) == -1)
{
   HttpSendSuccessfulResponse(hndl, HttpReturnString(HTTP_CONTENT_TYPE_HTML));
   HttpSendDataSprintf(hndl, "<HTML><HEAD><TITLE>DEMO Page 4
   <TITLE>%s</HEAD><BODY>", PORTAL_STYLE_SHEET);
   HttpSendDataSprintf(hndl, "<H2>This is demo page 4.
   </BODY>", PORTAL_STYLE_SHEET);
   break;
}

err = HTTP_STATUS_BAD_REQUEST;
break;

   case 0:// display server page
   HttpSendSuccessfulResponse(hndl,
   HttpReturnString(HTTP_CONTENT_TYPE_HTML));
   HttpSendDataSprintf(hndl, "<HTML><HEAD><TITLE>DEMO Page 1
   </TITLE>%s</HEAD><BODY>", PORTAL_STYLE_SHEET);
   HttpSendDataSprintf(hndl, "<H1>This is demo page 1.</H1>",
   PORTAL_STYLE_SHEET);

   HttpSendDataSprintf(hndl, "<B><A HREF=/%s/%s>Demo Page 2
   </A>\n", pzDemoService, pzDemoPage2+1);
   HttpSendDataSprintf(hndl, "<B><A HREF=/%s/%s>Demo Page
   </A>\n", pzDemoService, pzDemoPage3+1);
   HttpSendDataSprintf(hndl, "<B><A HREF=/%s/%s>Demo Page 4
   </A>\n", pzDemoService, pzDemoPage4+1);
   HttpSendDataSprintf(hndl, "</BODY>");
   break;

   default:// URL passed was bad - bail out
   err = HTTP_STATUS_BAD_REQUEST;
   break;
}
if(err)
   HttpSendErrorResponse(hndl, err);
else
   HttpEndDataResponse(hndl);

return(err);
}
```

2 NetWare Remote Manager Registration Functions

This section contains reference information for the functions that register and unregister service methods and health monitors. They are arranged alphabetically in the following groups:

- “Service Method Registration” on page 19
- “Health Monitoring Functions” on page 28

Service Method Registration

This section alphabetically lists the service method functions and describes their purpose, syntax, parameters, and return values.

DeRegisterFileSystemServiceMethod

De-registers the default file system method with the NetWare HTTP Stack.

Syntax

```
#include <httpexp.h>

BOOL DeRegisterFileSystemServiceMethod (
  void   *pRTag);
```

Parameters

pRTag

(IN) Points to a NetWare resource tag.

Return Values

Constant	Description
TRUE	The Service Method has been de-registered.
FALSE	The Service Method failed de-registration.

DeRegisterServiceMethod

Removes a named Service Method from the NetWare HTTP Stack.

Syntax

```
#include <httpexp.h>

BOOL DeRegisterServiceMethod (
  void  *pzServiceName,
  void  *pServiceTag,
  int    szServiceTagLength,
  UINT32 (*pServiceMethodCallback)(
              HINTERNET  hndl,
              void      *pExtraInfo,
              UINT32     szExtraInfo,
              UINT32     InformationBits),
  void   *pRTag,
  UINT32   pReturnCode);
```

Parameters

pzServiceName

(IN) Points to an ASCIIZ string describing the Service Method.

pServiceTag

(IN) Points to the ASCII string representing the method.

szServiceTagLength

(IN) Specifies the length in bytes of the Service Tag.

pServiceMethodCallback

(IN) Points to the function to invoke when the first component of a relative URL path is equivalent to the Service Tag. The callback method is passed the following parameters:

- hndl—(IN) Specifies the HTTP stack handle.
- pExtraInfo—(IN) Points to any data following the relative URL, or to the break character if encountered in the relative URL.
- szExtraInfo—(IN) Specifies the size of the pExtraInfo buffer, in bytes.
- InformationBits—(IN) See “Information Bits” on page 132.

pRTag

(IN) Points to a NetWare resource tag.

pReturnCode

(OUT) Points to where the error code is placed, if registration fails.

Return Values

Constant	Description
TRUE	The Service Method has been de-registered.
FALSE	The Service Method failed de-registration. For failure reason, see pReturnCode.

See Also

RegisterServiceMethod (page 24), RegisterServiceMethodEx (page 26)

RegisterFileSystemServiceMethodEx

Registers a default file system method with the NetWare HTTP Stack.

Syntax

```
#include <httpexp.h>

BOOL RegisterFileSystemServiceMethodEx (
   UINT32   (*pServiceMethodCallback)(
               HINTERNET   hndl,
               void       *pExtraInfo,
               UINT32      szExtraInfo,
               UINT32      InformationBits),
   void    *pRTag,
   UINT32   pReturnCode
   UINT32   FileSystemID,
   UINT32   optionFlags);
```

Parameters

pServiceMethodCallback

(IN) Points to the function to invoke when the first component of a relative URL path is equivalent to the Service Tag. The callback method is passed the following parameters:

- hndl—(IN) Specifies the HTTP stack handle.
- pExtraInfo—(IN) Points to any data following the relative URL, or to the break character if encountered in the relative URL.
- szExtraInfo—(IN) Specifies the size of the pExtraInfo buffer, in bytes.
- InformationBits—(IN) See "Information Bits" on page 132.

pRTag

(IN) Points to a NetWare resource tag.

pReturnCode

(I/O) Points to where the error code is placed, if registration fails.

FileSystemID

(IN) Specifies.

optionFlags

(IN) Specifies.

Return Values

Constant	Description
TRUE	The Service Method has been registered as the default file system method.
FALSE	The Service Method failed registration. For failure reason, see pReturnCode.

RegisterServiceMethod

Registers a callable method with the NetWare HTTP Stack, which is invoked when the tag is encountered at the beginning of the relative URL path.

Syntax

```
#include <httpexp.h>

BOOL RegisterServiceMethod (
  void  *pzServiceName,
  void  *pServiceTag,
  int    serviceTagLength,
  UINT32 (*pServiceMethodCheck)(
              HINTERNET  hndl,
              void      *pExtraInfo,
              UINT32     szExtraInfo,
              UINT32     InformationBits),
  void   *pRTag,
  UINT32 *pReturnCode);
```

Parameters

pzServiceName

(IN) Points to an ASCIIZ string describing the Service Method.

pServiceTag

(IN) Points to the ASCII string representing the method.

serviceTagLength

(IN) Specifies the length in bytes of the Service Tag.

pServiceMethodCheck

(IN) Points to the function to invoke when the first component of a relative URL path is equivalent to the Service Tag. The callback method is passed the following parameters:

- hndl—(IN) Specifies the HTTP stack handle.
- pExtraInfo—(IN) Points to any data following the relative URL, or to the break character if encountered in the relative URL.
- szExtraInfo—(IN) Specifies the size of the pExtraInfo buffer, in bytes.
- InformationBits—(IN) See “Information Bits” on page 132

pRTag

(IN) Points to a NetWare resource tag.

pReturnCode

(I/O) Points to where the error code is returned, if registration fails.

Return Values

Constant	Description
TRUE	The Service Method has been registered.
FALSE	The Service Method failed registration. For failure reason, see pReturnCode.

Remarks

RegisterServiceMethod registers the service method with a default of zero rights (the method handles rights check).

The first call to the registered method is an initialization call. The CONTROL_INITIALIZATION_BIT is set in the InformationBits to signal this call. This allows the method to do any of its own initialization it needs to do before handling HTTP requests. Also, when HTTPSTK unloads or the method is deregistered, another call to the method is made with the CONTROL_DEINITIALIZATION_BIT set. Make sure any symbols dynamically imported from HTTPSTK are released before returning from your method on deinitialization.

See Also

RegisterServiceMethodEx (page 26), DeRegisterServiceMethod (page 21)

RegisterServiceMethodEx

Register a callable method with the NetWare HTTP Stack, which is invoked when the tag is encountered at the beginning of the relative URL path.

Syntax

```
#include <httpexp.h>

BOOL RegisterServiceMethod (
  void   *pzServiceName,
  void   *pServiceTag,
  int     serviceTagLength,
  void   *pTableOfContentsStruc,
  UINT32  requestedRights,
  void   *pReserved,
  UINT32 (*pServiceMethod)(
              HINTERNET  hndl,
              void      *pExtraInfo,
              UINT32     szExtraInfo,
              UINT32     InformationBits),
  void   *pRTag,
  UINT32 *pReturnCode);
```

Parameters

pzServiceName

(IN) Points to an ASCIIZ string describing the Service Method.

pServiceTag

(IN) Points to the ASCII string representing the method.

serviceTagLength

(IN) Specifies the length in bytes of the Service Tag.

pTableOfContentsStruc

(IN) Points to an optional TOCregistration (page 138) structure which adds your entry to the table of contents on the entry page of NetWare Remote Manager. See also "Registering Your Service" on page 11.

requestedRights

(IN) Specifies the rights required for the service method to be called. If the method wants to handle all rights issues, a zero may be passed. For information on the rights settings, please refer to "Information Bits" on page 132.

pReserved

(IN) Reserved for future use. Must be set to zero.

pServiceMethod

(IN) Points to the function to invoke when the first component of a relative URL path is equivalent to the Service Tag. The callback method is passed the following parameters:

- hndl—(IN) Specifies the HTTP stack handle.

- pExtraInfo—(IN) Points to any data following the relative URL, or to the break character if encountered in the relative URL.
- szExtraInfo—(IN) Specifies the size of the pExtraInfo buffer, in bytes.
- InformationBits—(IN) See "Information Bits" on page 132.

pRTag

(IN) Points to a NetWare resource tag.

pReturnCode

(I/O) Points to where the error code is returned, if registration fails.

Return Values

Constant	Description
TRUE	The Service Method has been registered.
FALSE	The Service Method failed registration. For failure reason, see pReturnCode.

Remarks

The RegisterServiceMethodEx function allows a table of contents structure to be passed, allowing the registered method to appear in the table of contents on the front page of NetWare Remote Manager.

See Also

RegisterServiceMethod (page 24), DeRegisterServiceMethod (page 21)

Health Monitoring Functions

This set of functions is used to make server health information available in the Server Health listing in NetWare Remote Manager. Any NLM running on the server that would like to give the administrator and users more information about how their part of the system is running can register using this interface. This interface allows only the owner of the system to provide information. The information provided is updated by each subsystem at the interval that they desire.

This section alphabetically lists the health monitor functions and describes their purpose, syntax, parameters, and return values:

PostHealthState

Is a prototype for a function that updates the health of a component.

Syntax

```
#include <pexports.h>

UINT32 PostHealthState (
  UINT32   moduleHandle,
  UINT32   handle,
  char    *maxValueString,
  char    *peakValueString,
  char    *currentValueString,
  UINT8    currentState,
  UINT8   *resUINT8,
  UINT32   resUINT32);
```

Parameters

moduleHandle

(IN) Specifies the NLM handle that is passed to you at startup. This parameter ensures that only the module which registered this health monitor is posting data for it.

handle

(IN) Specifies the handle that was returned to you when you registered your health monitor

maxValueString

(IN) Points to a formatted string that describes your maximum value. This string is limited to 48 characters. If you want to leave a value the same, pass in a NULL pointer.

peakValueString

(IN) Points to a formatted string that describes your peak value. This string is limited to 48 characters. If you want to leave a value the same, pass in a NULL pointer.

currentValueString

(IN) Points to a formatted string that describes your current value. This string is limited to 48 characters. If you want to leave a value the same, pass in a NULL pointer.

currentState

(IN) Specifies the current health state with one of the following values:

Value	Name
0	GREENSTATUS
1	YELLOWSTATUS
2	REDSTATUS
10	INACTIVESTATUS
20	OFFLINE

resUINT8

(IN) Available for future expansion.

resUINT32

(IN) Available for future expansion.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code. See “Health Monitor Error Codes” on page 133.

Remarks

The PostHealthState function reports the health of your system. You can only call this function after you have registered as a health monitor.

If you receive the error code HEALTH_MONITOR_REPLACED, stop sending your health state because another NLM has replaced your health monitor. This usually happens only with health monitors that are registered by the Portal NLM. They are replaced by NLMs that are better able to determine a component’s health.

RegisterHealthMonitor

Is a prototype for a function that registers a health monitor.

Syntax

```
#include <pexports.h>

UINT32 RegisterHealthMonitor (
  UINT32   moduleHandle,
  UINT32  *handle,
  UINT8    category,
  UINT8    flags,
  char    *descriptionString,
  char    *URL,
  char    *helpURL,
  char    *oldDescriptionString);
```

Parameters

moduleHandle

(IN) Specifies the NLM handle that is passed to you at startup. The handle is used for resource tracking at unload time.

handle

(OUT) Points to a handle that is used to identify your health monitor when you unregister it or post health status.

category

(IN) Specifies the type monitor so that like monitors can be grouped together in the list of health monitors. For available categories, see “Categories” on page 133.

flags

(IN) Specifies whether or not this is a new health monitor or if you are replacing one that is already there. It can also indicate that you want your page to be displayed in the full browser window.

The following values are supported:

Value	Flag
0x01	NEW_ENTRY
0x02	REPLACE_ENTRY
0x04	FULL_PAGE_DISPLAY (Requires Portal 1.1)

descriptionString

(IN) Points to a string that will be displayed in the chart under the Description heading. This is the string that has a link to your more detailed information. If the description is a NULL and the oldDescriptionString matches the current description, the description stays the same.

URL

(IN) Points to a URL that points to your more detailed information when the user clicks on your Description string. It needs to be the full URL for your page.

helpURL

(IN) Points to the URL for your specific Health Help information, when the user clicks on your question mark icon for specific information. It needs to be the full URL for your help page.

oldDescriptionString

(IN) Points to the description string for the health monitor that you desire to replace. It is only valid when the flags parameter is set to REPLACE_ENTRY.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code. See "Health Monitor Error Codes" on page 133.

Remarks

The RegisterHealthMonitor function allows any NLM running on the server to register with the Portal NLM to indicate that they are monitoring something and would like to report its state. Each item registered receives a handle back, which is used to post a health status and to unregister the health monitor. Each registered health monitor is required to have a URL, help URL, description string flags, and a category.

RegisterServerHealthThreshold

Is a prototype for a function that registers thresholds for a component.

Syntax

```
#include <pexports.h>

UINT32 RegisterServerHealthThreshold (
  UINT32   handle,
  char    *thresholdURL,
  UINT32   category,
  void   (*ReturnCurrentValues)(
          char   *suspectLimit,
          char   *criticalLimit));
```

Parameters

handle

(IN) Specifies the handle that was returned to you when you registered your health monitor and identifies which health monitor this threshold callback and URL are associated with.

thresholdURL

(IN) Points to a formatted string that will be associated with the description for the specific health item. This URL is called when the specific threshold link is selected.

category

(IN) Specifies the category into which you grouped your health monitor. For available categories, see “Categories” on page 133.

ReturnCurrentValues

(IN) Points to a callback function pointer that returns the current yellow and red threshold values. The callback function has the following parameters.

- suspectLimit—(OUT) Points to the current yellow threshold value.
- criticalLimit—(OUT) Points to the current red threshold value.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code. See “Health Monitor Error Codes” on page 133.

Remarks

The RegisterServerHealthThreshold function registers a URL to be returned when the specific health item threshold description is selected. The caller is required to supply a callback routine that allows the portal server health engine to retrieve the current values. It is not be necessary to unregister this call because the resources used to track this information are returned when the health item is unregistered.

If you receive the error code THRESHOLD_ALREADY_SET, the threshold callback has already been set for that specific health item. Check your code for a handle error and make sure you are not registering a threshold callback twice for the same health item.

UnregisterHealthMonitor

Is a prototype function for unregistering a health monitor.

Syntax

```
#include <pexports.h>

UINT32 UnregisterHealthMonitor (
  UINT32   moduleHandle,
  UINT32   handle,
  UINT8    category);
```

Parameters

moduleHandle

(IN) Specifies the NLM handle that was passed to you at startup. This parameter ensures that only the module that registered the monitor can unregister it.

handle

(IN) Specifies the handle that was returned to you when you registered your health monitor.

category

(IN) Specifies the category into which you grouped your health monitor. For available categories, see “Categories” on page 133.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code. See “Health Monitor Error Codes” on page 133.

Remarks

The UnregisterHealthMonitor function allows you to unregister a previously registered health monitor. The function requires the moduleHandle for your NLM and the handle that was returned when the health monitor was registered.

3 HTTP Server Functions

This section contains reference information for the functions that allow you to accept an HTTP request and to send an HTTP response. They are arranged alphabetically in the following groups:

- "Request Header Functions" on page 37, which retrieve information from the request.
- "Response Header Functions" on page 55, which add items to the response header and send it.
- "Data Response Functions" on page 71, which add data to the response and send it.

For functions that can be used by both a server and a client application, see "HTTP Server and Client Functions" on page 111.

Request Header Functions

The request header functions include the following:

HttpActivateFileMode

Writes all output to a user file via the supplied file handle.

Syntax

```
#include <httpexp.h>

void HttpActivateFileMode (
  HINTERNET   hndl,
  UINT32      netwareFileHandle,
  UINT32      fileModeFlags);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

netwareFileHandle

(IN) Specifies the file handle of an user created file.

fileModeFlags

(IN) Specifies options on how the system deals with the file.

HttpCheckIfModifiedSinceTime

Compares the request message “If Modified Since” time against caller supplied time, and determines if they are equal.

Syntax

```
#include <httpexp.h>

int HttpCheckIfModifiedSinceTime (
  HINTERNET   hndl,
  int         InternetTime);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

InternetTime

(IN) Specifies the Internet Time to check against the “If Modified Time” in the HTTP request.

Return Values

Decimal	Description
0	The times are the same and no update is necessary.
1	The times are different and the object should be sent

HttpDeActivateFileMode

Closes an active file mode session.

Syntax

```
#include <httpexp.h>

void HttpDeActivateFileMode (
  HINTERNET   hndl);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

See Also

HttpActivateFileMode (page 38)

HttpExtractDestAddress

Returns the destination address of the request in the specified format.

Syntax

```
#include <httpexp.h>

int HttpExtractDestAddress (
   HINTERNET   hndl,
   int        *pDestIPAddr,
   int        *pDestIPPort,
   char       *pzDestIPAddrBuf,
   char       *pzDestPortBuf);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

pDestIPAddr

(OUT) If not NULL, points to a buffer for the IP address in binary format.

pDestIPPort

(OUT) If not NULL, points to a buffer for the port address in binary format.

pzDestIPAddrBuf

(OUT) If not NULL, points to a buffer for the IP address string in dotted decimal format.

pzDestPortBuf

(OUT) If not NULL, points to a buffer for the port address in decimal format.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
255	ERR_BAD_PARAMETER	The hndl parameter is invalid.

Remarks

You only need to provide pointers for the values you need. The other parameters can be set to NULL.

See Also

HttpExtractSrcAddress (page 42)

HttpExtractSrcAddress

Returns the source address of the request in the specified format.

Syntax

```
#include <httpexp.h>

int HttpExtractSrcAddress (
   HINTERNET   hndl,
   int        *pSrcIPAddr,
   int        *pSrcIPPort,
   char       *pzSrcIPAddrBuf,
   char       *pzSrcPortBuf);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

pSrcIPAddr

(OUT) If not NULL, points to a buffer for the IP address in binary format.

pSrcIPPort

(OUT) If not NULL, points to a buffer for the port address in binary format.

pzSrcIPAddrBuf

(OUT) If not NULL, points to a buffer for the IP address string in dotted decimal format.

pzSrcPortBuf

(IN) If not NULL, points to a buffer for the port address in decimal format.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
255	ERR_BAD_PARAMETER	The hndl parameter is invalid.

Remarks

You only need to provide pointers for the values you need. The other parameters can be set to NULL.

See Also

HttpExtractDestAddress (page 41)

HttpGetContext

Returns the context handle that was associated with the HINTERNET handle.

Syntax

```
#include <httpexp.h>

int HttpGetContext (
  HINTERNET    hndl,
  UINT32_PTR  *pContext);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

pContext

(OUT) Points to where the context pointer is placed.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	Context is not set.
255	ERR_BAD_PARAMETER	Invalid handle.

See Also

HttpSetContext (page 52).

HttpReturnHeaderVersion

Returns the major and minor versions from the HTTP request.

Syntax

```
#include <httpexp.h>

BOOL HttpReturnHeaderVersion (
  HINTERNET    hndl,
  UINT32_PTR   lpHttpMajorVersion,
  UINT32_PTR   lpHttpMinorVersion);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

lpHttpMajorVersion

(OUT) Points to where the request header major version number is placed.

lpHttpMinorVersion

(OUT) Points to where the request header minor version number is placed.

Return Values

Constant	Description
TRUE	Success.
FALSE	The function failed to return the version information.

HttpReturnHttpString

Returns a pointer to the string "http" if the connection is not associated with SSL and to the string "https" if the connection is an SSL connection.

Syntax

```
#include <httpexp.h>

char *HttpReturnHttpString (
   HINTERNET   hndl);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

Return Values

If successful, returns a pointer to the “http” or “https” string in the request header. Otherwise, returns 0.

See Also

HttpReturnString (page 121)

HttpReturnLoginServiceTagString

Returns the Login Service Method Tag.

Syntax

```
#include <httpexp.h>

UINT32 HttpReturnLoginServiceTagString (
   HINTERNET   hndl,
   char       *buffer,
   UINT32      size);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

buffer

(OUT) Points to where the Login Service Method Tag is placed.

size

(OUT) Specifies, in bytes, the size of the buffer.

Return Values

If successful, returns HTTP_SUCCESSFUL. Otherwise, returns HTTP_BAD_REQUEST, indicating that the buffer is too small for the tag information.

HttpReturnPathBuffers

Returns the addresses of the path buffers and their size.

Syntax

```
#include <httpexp.h>

int HttpReturnPathBuffers (
  HINTERNET    hndl,
  UINT32_PTR   lpszBufs,
  char       **path,
  char       **cnvpath);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

lpszBufs

(OUT) Points to where the maximum size of the path buffers is placed.

path

(OUT) Points to where the address of the raw path buffer is placed. The raw path buffer contains the path portion of the URL.

cnvpath

(OUT) Points to where the address of the un-escaped path buffer is placed.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
255	ERR_BAD_PARAMETER	Invalid handle.

HttpReturnPostDataBuffer

Returns the address and length of the received Post Data buffer.

Syntax

```
#include <httpexp.h>

int HttpReturnPostDataBuffer (
  HINTERNET    hndl,
  char       **pPostDataBuffer,
  UINT32_PTR   lpPostDataBuffer);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

pPostDataBuffer

(OUT) Points to where a pointer to the Post Data Buffer is placed.

lpPostDataBuffer

(OUT) Points to where the length of the Post Data Buffer, not including terminating NULL, is placed.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
121	ERR_NOT_ITEMS_FOUND	No post data available.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpReturnPutDataBuffer

Returns the address and length of the received Post Data buffer.

Syntax

```
#include <httpexp.h>

int HttpReturnPutDataBuffer (
  HINTERNET    hndl,
  UINT32_PTR   lpTotalFileSize,
  UINT32_PTR   lpCurrentFileOffset,
  char       **pPutDataBuffer,
  UINT32_PTR   lpPutDataLen);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

lpTotalFileSize

(OUT) Points to where the total file size is placed.

lpCurrentFileOffset

(OUT) Points to where the current file offset is placed.

pPutDataBuffer

(IN/OUT) Points to where to place a pointer to the received Put Data Buffer.

lpPutDataLen

(IN/OUT) Points to where to place the length of the received Put Data Buffer.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
121	ERR_NOT_ITEMS_FOUND	No Put data available.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpReturnRawRequest

Places the raw incoming request URL in the user's buffer.

Syntax

```
#include <httpexp.h>

int HttpReturnRawRequest (
   HINTERNET   hndl,
   UINT32      szRawRequestBuffer,
   char       *pzRawRequestBuffer);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

szRawRequestBuffer

(IN) Specifies the size of the buffer.

pzRawRequestBuffer

(OUT) Points to the buffer to place the raw request.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
119	ERR_BUFFER_TOO_SMALL	The raw request buffer is too small.
255	ERR_BAD_PARAMETER	A bad parameter was passed to the function.

See Also

HttpReturnString (page 121)

HttpReturnRequestMethod

Returns the numerical value of the request method.

Syntax

```
#include <httpexp.h>

BOOL HttpReturnRequestMethod (
  HINTERNET    hndl,
  UINT32_PTR   httpRequestMethodType);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

httpRequestMethodType

(OUT) Points to where the numerical equivalent of the request method text string is placed. For possible values, see "Request Method Types" on page 131.

Return Values

Constant	Description
TRUE	Success.
FALSE	Failure; the handle is invalid.

HttpSetContext

Associates a user-defined context handle with the HINTERNET handle.

Syntax

```
#include <httpexp.h>

int HttpSetContext (
  HINTERNET    hndl,
  UINT32_PTR   pContext);
```

Parameters

hndl

(IN) Specifies an HINTERNET handle.

pContext

(IN) Specifies the local context.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
255	ERR_BAD_PARAMETER	Invalid handle.

See Also

HttpGetContext (page 43).

HttpStackIpInformation

Provides a means of determining the IP addresses that the NetWare HTTP Stack is listening on.

Syntax

```
#include <httpexp.h>

BOOL HttpStackIpInformation (
  UINT32      infoLevel,
  void       *infoBuffer,
  UINT32_PTR  numberOfIpAddresses,
  void      **addrInfo,
  UINT32_PTR  listenPortNumber);
```

Parameters

infoLevel

(IN) Reserved for future use. Value must be set to zero.

infoBuffer

(IN) Reserved for future use. Address must be set to NULL.

numberOfIpAddresses

(OUT) Address where to place the UINT32 number of IP addresses the HTTP Stack is listening on.

addrInfo

(OUT) Address where to place the address of the UINT32 table containing the IP addresses.

listenPortNumber

(OUT) Address where to place the UINT32 Listen Port number.

Return Values

Constant	Description
TRUE	Success.
FALSE	Failure; most likely cause is bad parameters.

serviceGetConnectionNumber

Returns a NetWare connection number and task number from the HINTERNT handle.

Syntax

```
#include <httpexp.h>

int serviceGetConnectionNumber (
  HINTERNET    hndl,
  UINT32_PTR   lpNetWareConnectionNumber,
  UINT32_PTR   lpNetWareTaskNumber);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

lpNetWareConnectionNumber

(OUT) Points to where the NetWare connection number is placed.

lpNetWareTaskNumber

(OUT) Points to where the NetWare connection task number is placed.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
255	ERR_BAD_PARAMETER	Invalid handle.

Response Header Functions

Header response functions include the following functions:

HttpAddResponseHeaderContentLengthTag

Appends to an HTTP Response Header a content length attribute.

Syntax

```
#include <httpexp.h>

int HttpAddResponseHeaderContentLengthTag (
  HINTERNET    hndl,
  UINT32       FileSizeHigh,
  UINT32       FileSizeLow);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

FileSizeHigh

(IN) Specifies the value of the upper 32 bits of a 64-bit file size.

FileSizeLow

(IN) Specifies the value of the lower 32 bits of a 64-bit file size.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader function has already been called.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpAddResponseHeaderContentTypeTag

Appends a content type attribute to an HTTP Response Header and returns a flag indicating if a content length attribute is needed.

Syntax

```
#include <httpexp.h>

int HttpAddResponseHeaderContentTypeTag (
  HINTERNET    hndl,
  void        *pzExtName,
  int          szExtName,
  UINT32_PTR   pContentLengthFlag);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

pzExtName

(IN) Points to a file name extension.

szExtName

(IN) Specifies, in bytes, the size of the file name extension.

pContentLengthFlag

(OUT) Points to a flag which, on successful completion, indicates whether the Content Length attribute is required:

- TRUE—Content Length attribute is required.
- FALSE—Content Length attribute is not required.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader function has already been called.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpAddResponseHeaderDateTag

Appends a Date-Time attribute to an HTTP Response Header.

Syntax

```
#include <httpexp.h>

int HttpAddResponseHeaderDateTag (
  HINTERNET    hndl);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader function has already been called.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpAddResponseHeaderLastModifiedTag

Appends a Last Modified attribute to an HTTP Response Header using the provided time.

Syntax

```
#include <httpexp.h>

int HttpAddResponseHeaderLastModifiedTag (
  HINTERNET    hndl,
  UINT32       gmtTimeInSeconds);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

gmtTimeInSeconds

(IN) Specifies GMT, in seconds, for the Last Modified Header Attribute.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader function has already been called.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpAddResponseHeaderLocationTag

Appends a Location attribute to an HTTP Response Header using the provided URL.

Syntax

```
#include <httpexp.h>

int HttpAddResponseHeaderLocationTag (
  HINTERNET    hndl,
  char        *pzURL);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

pzURL

(IN) Points to an ASCIIZ URL string.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader function has already been called.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpAddResponseHeaderSetCookieTag

Appends an ASCIIZ string containing a caller supplied cookie to an HTTP Response Header.

Syntax

```
#include <httpexp.h>

int HttpAddResponseHeaderSetCookieTag (
  HINTERNET    hndl,
  void        *pCookie,
  UINT32       szCookie);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

pCookie

(IN) Points to an ASCIIZ Cookie string.

szCookie

(IN) Specifies the size of the Cookie string.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader function has already been called.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpAddResponseHeaderStringContentType

Appends a content-type attribute to an HTTP Response Header with the provided string.

Syntax

```
#include <httpexp.h>

int HttpAddResponseHeaderSetCookieTag (
  HINTERNET    hndl,
  void        *pzContentType);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

pzContentType

(IN) Points to an ASCIIZ Content Type Header attribute (assumed to include the CR LF sequence).

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader function has already been called.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpAddResponseHeaderTags

Appends the provided string to an HTTP Response Header.

Syntax

```
#include <httpexp.h>

int HttpAddResponseHeaderTags (
  HINTERNET    hndl,
  void        *pHeaders,
  UINT32       dwHeadersLength);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

pHeaders

(IN) Points to a Header Attribute or Header Attributes (separated by CR LF).

dwHeadersLength

(IN) Specifies the size of the Header Attribute or Header Attributes.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader function has already been called.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpEndDataResponse

Flushes any remaining data in the Tx buffer and then closes the Data Response session.

Syntax

```
#include <httpexp.h>

int HttpEndDataResponse (
  HINTERNET    hndl);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader function must be completed before data functions can be used.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpOpenResponseHeaderTag

Opens a Response Message Header session.

Syntax

```
#include <httpexp.h>

int HttpOpenResponseHeaderTag (
  HINTERNET    hndl,
  UINT32       HTTP_ERROR_CODE);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

HTTP_ERROR_CODE

(OUT) Specifies a value which contains the HTTP Status Code to use. See "HTTP Status Codes" on page 125.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader function has already been called.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpSendErrorPackageResponse

Sends an HTTP error code and an HTML reply.

Syntax

```
#include <httpexp.h>

int HttpSendErrorPackageResponse (
   HINTERNET   hndl,
   UINT32      HTTP_ERROR_CODE,
   void       *pData,
   UINT32      szData);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

HTTP_ERROR_CODE

(IN) Specifies the error code to send. For possible values, see “HTTP Status Codes” on page 125.

pData

(IN) Points to the data to send.

szData

(IN) Specifies the size of the data to send.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
153	ERR_DIRECTORY_FULL	The call was made out of sequence.
168	ERR_ACCESS_DENIED	The call was made out of sequence. The response has already been sent.
255	ERR_BAD_PARAMETER	A bad parameter was passed to the function.

Remarks

The HttpSendErrorPackageResponse function allows you to reply with a single call when an error condition occurs.

See Also

HttpSendPackage (page 68)

HttpSendErrorResponse

Sends a complete Header & Data session back to the client with the supplied HTTP status code.

Syntax

```
#include <httpexp.h>

int HttpSendErrorResponse (
  HINTERNET    hndl,
  UINT32       HTTP_ERROR_CODE);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

HTTP_ERROR_CODE

(IN) Specifies a value which contains the HTTP Status Code to use. See "HTTP Status Codes" on page 125.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
255	ERR_BAD_PARAMETER	Invalid handle.

HttpSendPackage

Sends a complete response message (including Header and Data) to the client using the data and data size provided.

Syntax

```
#include <httpexp.h>

int HttpSendPackage (
  HINTERNET    hndl,
  void        *pData,
  UINT32       szData);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

pData

(IN) Points to the HTML data to send.

szData

(IN) Specifies the size of data to send.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
255	ERR_BAD_PARAMETER	Invalid handle.

Remarks

The HttpSendPackage function allows you to reply with a single call when HTTP status of your reply is 200 (HTTP_STATUS_OK) and any data you need to send is in HTML format.

See Also

HttpSendErrorPackageResponse (page 66)

HttpSendResponseHeader

Closes an already-opened Response Header session and opens the Data Response Session.

Syntax

```
#include <httpexp.h>

int HttpSendResponseHeader (
  HINTERNET    hndl);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader function has already been called.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpSendSuccessfulResponse

Opens a response Header session with an HTTP_STATUS_OK status, current Internet date and time attribute, and the Content Type attribute string provided by the caller.

Syntax

```
#include <httpexp.h>

int HttpSendSuccessfulResponse (
  HINTERNET    hndl,
  void        *pzContentType);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

pzContentType

(IN) Points to a Header Content Type ASCIIZ string.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
255	ERR_BAD_PARAMETER	Invalid handle.

Data Response Functions

Data response functions include the following functions:

BuildAndSendFooter

Closes the BODY and the HTML tags.

Syntax

```
#include <pexports.h>

UINT32 BuildAndSendFooter (
  HINTERNET   hndl);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

Return Values

Return call from HttpSendDataSprintf.

BuildAndSendHeader

Opens a consistent header for all pages.

Syntax

```
#include <pexports.h>

UINT BuildAndSendHeader (
  HINTERNET   handle,
  char       *windowTitle,
  char       *pageIdentifier,
  char        refresh,
  UINT32      refreshDelay,
  UINT        flags,
  void        (*AddHeaderText)(HINTERNET handle),
  char       *bodyTagText
  char       *helpURL);
```

Parameters

handle

(IN) Specifies the HTTP stack handle.

windowTitle

(IN) Points to the title of the page. If this parameter is NULL, a default window title is provided.

pageIdentifier

(IN) Points to the text that is printed out on the header as a title for the page. If this parameter is NULL, a default window title is provided.

refresh

(IN) Specifies whether the page is to be refreshed:

- 0 = No refresh.
- Other = The page is set to automatically refresh itself.

refreshDelay

(IN) Specifies the number of seconds that should elapse before refreshing the page.

flags

(IN) Reserved for future use.

AddHeaderText

(IN) If this is not NULL, points to a function that puts out information in the header portion of the page. This call function is passed the following parameters:

- handle—(IN) Specifies the HTTP stack handle.

bodyTagText

(IN) Points to additional text that is put in the BODY tag. This is necessary for some functionality. For example, you may put in <BODY onLoad="SomeJavaScriptFunction()"> which will call the SomeJavaScriptFunction when the page has loaded.

helpURL

(IN) Points to the URL of a page that contains help information for the current page.

Return Values

If successful, returns 0.

Remarks

This function can sleep.

BuildAndSendUpdateHeader

Opens a consistent header for all pages.

Syntax

```
#include <pexports.h>

UINT BuildAndSendUpdateHeader (
  HINTERNET  handle,
  void      *info,
  UINT32     infoLen,
  char      *windowTitle,
  char      *pageIdentifier,
  char       Refresh,
  UINT32     refreshDelay,
  UINT       flags,
  void       (*AddHeaderText)(HINTERNET handle),
  char      *bodyTagText,
  char      *pageToRefresh,
  char      *helpURL);
```

Parameters

handle

(IN) Specifies the HTTP stack handle.

info

(IN) Points to the title of the page. If this parameter is NULL, a default window title is provided.

infoLen

(IN) Specifies the length, in bytes, of the info parameter.

windowTitle

(IN) Points to the title of the page. If this parameter is NULL, a default window title is provided.

pageIdentifier

(IN) Points to the text that gets printed out on the header as a title for the page. If this parameter is NULL, a default window title is provided.

refresh

(IN) Specifies whether the page is to be refreshed:

- 0 = No refresh.
- Other = The page is set to automatically refresh itself.

refreshDelay

(IN) Specifies the number of seconds that should elapse before refreshing the page.

flags

(IN) Reserved for future use.

AddHeaderText

(IN) If this is not NULL, points to a function that puts out information in the header portion of the page. This call function is passed the following parameters:

- handle—(IN) Specifies the HTTP stack handle.

bodyTagText

(IN) Points to additional text that is put in the BODY tag. This is necessary for some functionality. For example, you may put in <BODY onLoad="SomeJavaScriptFunction()"> which will call the SomeJavaScriptFunction when the page has loaded.

pageToRefresh

(IN) Points to the URL of the page to refresh.

helpURL

(IN) Points to the URL of a page that contains help information for the current page.

Return Values

If successful, returns 0.

BuildAndSendHelpHeader

Opens a consistent header for all help pages.

Syntax

```
#include <pexports.h>

UINT BuildAndSendHelpHeader (
  HINTERNET  handle,
  char      *windowTitle,
  UINT       flags,
  void       (*AddHeaderText)(HINTERNET handle),
  char      *BodyTagText);
```

Parameters

handle

(IN) Specifies the HTTP stack handle.

windowTitle

(IN) Points to the title of the page. If this parameter is NULL, a default window title is provided.

flags

(IN) Reserved for future use.

AddHeaderText

(IN) If this is not NULL, points to a function that puts out information in the header portion of the page. This call function is passed the following parameters:

- handle—(IN) Specifies the HTTP stack handle.

bodyTagText

(IN) Points to additional text that we will put in the BODY tag. This is necessary for some functionality. For example, you may put in <BODY onLoad="SomeJavaScriptFunction()"> which will call the SomeJavaScriptFunction when the page has loaded.

Return Values

If successful, returns 0.

HttpSendDataFlush

Sends the data in the Tx buffer immediately.

Syntax

```
#include <httpexp.h>

int HttpSendDataFlush (
  HINTERNET    hndl);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
162	ERR_IO_LOCKED	The application must complete the Header response before using data functions.
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader must be completed before data functions are used.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpSendDataResponse

Closes the Data Response session and flushes the data in the Tx buffer.

Syntax

```
#include <httpexp.h>

int HttpSendDataResponse (
  HINTERNET    hndl,
  void        *pBuffer,
  UINT32       szBuffer);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

pBuffer

(IN) Points to the data to be sent.

szBuffer

(IN) Specifies the size of data to be sent.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
162	ERR_IO_LOCKED	The application must complete the Header response before using data functions.
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader must be completed before data functions are used.
255	ERR_BAD_PARAMETER	Invalid handle.

HttpSendDataResponseNoCopy

Sends data to a client without copying the data to a local buffer.

Syntax

```
#include <httpexp.h>

int HttpSendDataResponseNoCopy (
   HINTERNET   hRequest,
   void       *pBuffer,
   UINT32      szBuffer);
```

Parameters

hRequest

(IN) Specifies the HTTP stack handle.

pBuffer

(IN) Points to the buffer containing the data to send.

szBuffer

(IN) Specifies the size of the data buffer.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader must be completed before data functions are used.
255	ERR_BAD_PARAMETER	Invalid handle.

Remarks

The HttpSendDataResponseNoCopy function is similar to the HttpSendDataResponse function but it does not copy the user's data to the local buffer before transmitting it. Because the data is not copied to the local buffer, this function should only be used when large amounts of data (over 4096 bytes per call) are being sent.

See Also

HttpSendDataResponse (page 79)

HttpSendDataTxBuffer

Sends the data to the Tx buffer provided by the caller.

Syntax

```
#include <httpexp.h>

int HttpSendDataTxBuffer (
  HINTERNET    hndl,
  UINT32       bytesToSend);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle.

bytesToSend

(IN) Specifies the number of bytes of data to send from the Tx buffer.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
162	ERR_IO_LOCKED	The application must complete the Header response before data functions can be used.
168	ERR_ACCESS_DENIED	The HttpSendResponseHeader function must be completed before data functions can be used.
255	ERR_BAD_PARAMETER	Invalid handle.

4 HTTP Client Services

The HTTP client interfaces allow NLMs to make HTTP requests and receive replies to those requests. These functions are available on NetWare 6.5 or later. This section describes the following:

HTTP Client Services Concepts

An HTTP client is an object that sends HTTP requests and receives HTTP replies. This section describes the following features of the HTTP client interfaces:

Requirements

NLMs using the HTTP client functions have the following requirements:

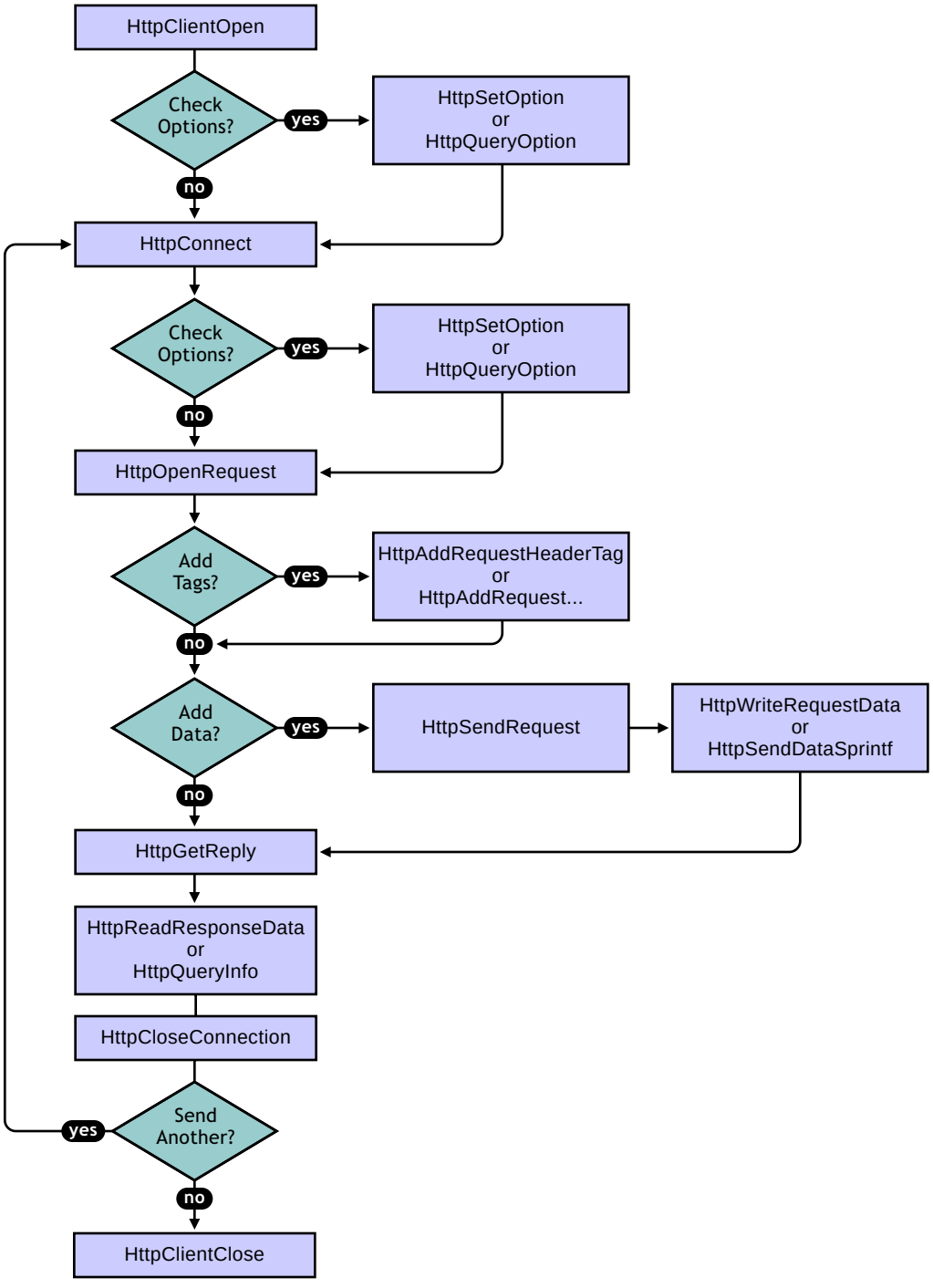
- NetWare 6.5 or later
- CLib or LibC functions to manage the NetWare resource tags
- The nwerror.h file from CLib for NetWare server error codes

HTTP Resources

The NetWare Remote Manager follows the HTTP/1.1 specification. For more information about this protocol and its specification, see Hypertext Transfer Protocol — HTTP/1.1 (http://www.w3.org/Protocols/rfc2616/rfc2616.html).

Granular Requests

A granular request gives you complete control of the content of the request header, body data, and response handling. This process is illustrated in the following graphic and then explained in the following paragraphs.



Starting the request. All granular requests start with calling the following functions in the order listed:

HttpClientOpen
HttpConnect
HttpOpenRequest

After HttpClientOpen or HttpConnect, you can call HttpSetOption to set the connection timeout for the connection or globally for the client, but this isn't required. You can also call HttpQueryOption to determine the current connection timeout for the handle.

What you call after HttpOpenRequest depends upon whether the request is complete or whether you need to add header tags or data. If the request is complete, you call the following function:

HttpGetReply

Adding to the request. If you have additional header tags to add after calling HttpOpenRequest, you call HttpAddRequestHeaderTag or one of the specialized header tag functions. If you have no data to add, you call the following function:

HttpGetReply

If you have data to add to the request, you call the following functions after you have added your request header tags:

HttpSendRequest
HttpWriteRequestData or HttpSendDataSprintf
HttpGetReply

Handling the reply. You call the following functions when the reply is received:

HttpQueryInfo or HttpReadResponseData
HttpCloseConnection
HttpClientClose

If you have additional requests, the client does not need to be closed, just the connection. You can reuse the client handle to open the connection again to send another request. However, HttpCloseConnection must be called after a reply has been received and the information digested. If the process errors out before receiving a reply, the HttpCloseConnection must be called to release the handle allocated with the HttpConnect call.

Synchronous or Asynchronous Mode

The HTTP client functions that actually send the request and receive the reply (such as HttpGetReply) can be used synchronously or asynchronously. If used synchronously, the function blocks until the reply is received. The synchronous mode should be used when sending a single request at a time and when blocking (sometimes for a long time for a busy of downed server) is not a problem. If used asynchronously, the function returns immediately with a return code that indicates that the request has been sent or that an error occurred. If an error occurred, the callback function is never called. If the request was handled successfully and sent, the callback function is called when the reply returns.

If your application handles lots of connections, you need to use the asynchronous mode of the functions. An application which handles multiple connections should not be designed to have a dedicated thread for each connection. Such a design uses too many resources. The asynchronous mode allows you to share the threads among the connections, and these functions provide a pContext parameter for you to pass any type of context data that you need to resume processing the reply.

The following functions support both a synchronous and an asynchronous mode:

HttpGetReply (page 97)
HttpSendRequest (page 105)
HttpWriteRequestData (page 108)

Header Request Tag Functions

Currently, the HTTP client interface supports only a few functions for adding tags to the request header. Most tags can be added by using the generic function, HttpAddRequestHeaderTag (page 89). The following are available to add specialized tags:

HttpAddRequestHeaderAuthorizationBasic (page 87)
HttpAddRequestHeaderContentLengthTag (page 88)
HttpReturnHttpString (page 45)

HTTP Client Functions

The HTTP client functions allow you to send an HTTP request and receive a reply. This section alphabetically lists these functions from the httpclnt.h file and describes their purpose, syntax, parameters, and return values.

For functions that can be used by both a server and a client application, see “HTTP Server and Client Functions” on page 111.

HttpAddRequestHeaderAuthorizationBasic

Adds the “Authorization: Basic” header to the HTTP request.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpAddRequestHeaderAuthorizationBasic (
   HINTERNET   hndl,
   char       *pzUsername,
   char       *pzPassword);
```

Parameters

hndl

(IN) Specifies the connection handle returned from HttpConnect.

pzUsername

(IN) Points to an ASCIIZ string containing the user’s name.

pzPassword

(IN) Points to an ASCIIZ string containing the user’s password.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
1	ERR_INSUFFICIENT_SPACE	Insufficient space in the response header.
168	ERR_ACCESS_DENIED	The session is not opened.
255	ERR_BAD_PARAMETER	An invalid parameter was passed to the function.

Remarks

The HttpAddRequestHeaderAuthorizationBasic function uuencodes the name and password and adds them to the end of the tag.

WARNING: This function should be used only on secure connections because uuencoding is not a form of encryption, and the username and password are easily extracted from unencrypted packets.

See Also

HttpAddRequestHeaderContentLengthTag (page 88)
HttpAddRequestHeaderTag (page 89)
HttpReturnHttpString (page 45)

HttpAddRequestHeaderContentLengthTag

Adds the content length header tag to the HTTP header.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpAddRequestHeaderContentLengthTag (
   HINTERNET   hndl,
   UINT32      FileSizeHigh,
   UINT32      FileSizeLow);
```

Parameters

hndl

(IN) Specifies the connection handle returned from HttpConnect.

FileSizeHigh

(IN) Specifies the most significant 32-bits of the 64-bit length in the body of the request.

FileSizeLow

(IN) Specifies the least significant 32-bits of the 64-bit length in the body of the request.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
1	ERR_INSUFFICIENT_SPACE	Insufficient space in the response header.
168	ERR_ACCESS_DENIED	The session is not opened.
255	ERR_BAD_PARAMETER	An invalid parameter was passed to the function.

See Also

HttpAddRequestHeaderAuthorizationBasic (page 87)
HttpAddRequestHeaderTag (page 89)
HttpReturnHttpString (page 45)

HttpAddRequestHeaderTag

Adds header tags to the request.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpAddRequestHeaderTag (
   HINTERNET   hndl,
   void       *pHeaders,
   UINT32      szHeadersLength);
```

Parameters

hndl

(IN) Specifies the connection handle returned from HttpConnect.

pHeaders

(IN) Points to an ASCIIZ string that is a complete HTTP header line, including the carriage-return and line-feed at the end of the line.

szHeadersLength

(IN) Specifies, in bytes, the length of the pHeaders string.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
1	ERR_INSUFFICIENT_SPACE	Insufficient space in the response header.
168	ERR_ACCESS_DENIED	The session is not opened.
255	ERR_BAD_PARAMETER	An invalid parameter was passed to the function.

Remarks

You use the HttpAddRequestHeaderTag function to add tags that do not have specialized header tag functions.

See Also

HttpAddRequestHeaderAuthorizationBasic (page 87)
HttpAddRequestHeaderContentLengthTag (page 88)
HttpReturnHttpString (page 45)

HttpAddRequestHeaderTransferEncodingChunked

Adds a transfer encoding chunked tag to the header.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpAddRequestHeaderTransferEncodingChunked (
   HINTERNET   hndl);
```

Parameters

hndl

(IN) Specifies the connection handle returned from HttpConnect.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
1	ERR_INSUFFICIENT_SPACE	Insufficient space in the response header.
168	ERR_ACCESS_DENIED	The session is not opened.
255	ERR_BAD_PARAMETER	An invalid parameter was passed to the function.

Remarks

The HttpAddRequestHeaderTransferEncodingChunked function must be called after a request has been opened (with HttpOpenRequest) but before the request is sent.

See Also

HttpAddRequestHeaderAuthorizationBasic (page 87)
HttpAddRequestHeaderContentLengthTag (page 88)
HttpAddRequestHeaderTag (page 89)

HttpClientOpen

Initializes the HTTP client interface.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

HINTERNET HttpClientOpen (
   char                          *appName,
   struct proxyInfoStructure     *pProxyInfo,
   UINT32                         dwFlags,
   UINT32                        (*pServiceFunction)(
                                    HINTERNET   hndl,
                                    void       *pRes,
                                    UINT32      szRes,
                                    UINT32      informationBits),
   void                           *pReserved,
   void                           *pRTag,
   UINT32                         *pFailureReasonCode);
```

Parameters

appName

(IN) Points an ASCIIZ string, which contains the name of your application.

pProxyInfo

(IN) Reserved for future use. Set to NULL.

dwFlags

(IN) Reserved for future use. Set to 0.

pServiceFunction

(IN) Points to the function to invoke when the HTTP client detects a situation that requires the user's attention. This function is passed the following parameters:

- hndl—Specifies the stack handle returned from the HttpClientOpen function.
- pRes—Reserved for future use. Returns NULL.
- szRes—Reserved for future use. Returns 0.
- informationBits—Reserved for future use.

pReserved

(IN) Reserved for future use. Set to NULL.

pRTag

(IN) Points to a NetWare resource tag, allocated with a signature of NetWareHttpClientResourceSignature.

pFailureReasonCode

(OUT) Points to an error code, if the open fails. If the open is successful, returns 0.

Return Values

If successful, returns an HTTP stack handle. Otherwise, returns NULL.

On failure, the pFailureReasonCode parameter returns one of the following values.

Decimal	Name	Description
150	ERR_NO_ALLOC_SPACE	Insufficient memory to perform the operation.
255	ERR_BAD_PARAMETER	The NetWare resource tag has an invalid signature.

Remarks

You must call the HttpClientOpen function once in your application before using any of the other functions in this interface. You use the handle returned by this function to open connections and to close the interface before your application quits.

See Also

HttpClientClose (page 93)

HttpClientClose

Closes the HTTP client interface.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpClientClose (
   HINTERNET   hndl,
   char       *appName,
   void       *pRTag,
   UINT32      dwFlags);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle returned by the HttpClientOpen function.

appName

(IN) Points to an ASCIIZ string, which contains the name of your application.

pRTag

(IN) Points to the NetWare resource tag that you used to open the interface.

dwFlags

(IN) Reserved for future use. Set to 0.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
255	ERR_BAD_PARAMETER	The NetWare resource tag has an invalid signature.

Remarks

Before closing the HTTP client interface, you must close all client connection handles by calling the HttpCloseConnection function.

See Also

HttpClientOpen (page 91)
HttpCloseConnection (page 94)

HttpCloseConnection

Closes a request connection.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpCloseConnection (
   HINTERNET   hndl);
```

Parameters

hndl

(IN) Specifies the connection handle to close.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
255	ERR_BAD_PARAMETER	The hndl parameter is not valid.

Remarks

The HttpCloseConnection function must be called once for every successful HttpConnect call. The function must be called even when subsequent calls after the HttpConnect call generate errors. To free all resources, it must be followed by an HttpClientClose call.

See Also

HttpConnect (page 95)
HttpClientClose (page 93)

HttpConnect

Starts a new HTTP connection to the specified server.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

HINTERNET HttpConnect (
   HINTERNET   hndl,
   char       *pServerName,
   UINT32      serverPort,
   UINT32      flags,
   UINT32     *pFailureReasonCode);
```

Parameters

hndl

(IN) Specifies the HTTP stack handle returned by the HttpClientOpen function.

pServerName

(IN) Points to an ASCIIZ string of the server name in either DNS or IP address format.

serverPort

(IN) Specifies the port number of the remote server. This number must be an unsigned integer.

flags

(IN) Reserved for future use. Set to 0.

pFailureReasonCode

(OUT) Points to an error code, if the connection fails to start. If the connection start is successful, returns 0.

Return Values

If successful, returns an HTTP client connection handle. Otherwise, returns NULL.

On failure, the pFailureReasonCode parameter returns one of the following values:

Decimal	Name	Description
1	ERR_INSUFFICIENT_SPACE	Insufficient memory to perform the operation.
168	ERR_ACCESS_DENIED	Insufficient permission to perform the operation.
255	ERR_BAD_PARAMETER	An invalid parameter was passed to the function.

Remarks

The HttpConnect function validates the connection information. This information, along with some other information, is stored in the handle. The actual connection is not made until an HttpSendRequest or HttpGetReply call is made.

The HttpConnect function must be followed by an HttpOpenRequest call, which allows you to format the request.

See Also

HttpClientOpen (page 91)
HttpGetReply (page 97)
HttpOpenRequest (page 99)
HttpSendRequest (page 105)

HttpGetReply

Receives and parses an HTTP reply header.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpGetReply (
   HINTERNET   hndl,
   int        *pHttpStatusCode,
   void       *pContext,
   void      (*pCallBackRoutine)(
               void       *pContext,
               HINTERNET   hndl,
               int         returnCode,
               int         httpStatusCode,
               void       *reserved)
   );
```

Parameters

hndl

(IN) Specifies the connection handle returned from HttpConnect.

pHttpStatusCode

(OUT) Points to HTTP reply status, parsed from the first line of the HTTP reply header. Only valid if no error occurs. For a list of possible values, see "HTTP Status Codes" on page 125.

pContext

(IN) Points to optional context data, which is passed to your callback function. For a synchronous request, set it to NULL.

pCallBackRoutine

(IN) Points to the callback function that is called when an asynchronous request returns. For a synchronous request, set it to NULL.

The callback function is passed the following parameters:

- pContext—Points to any optional context data that you passed in when you called HttpGetReply.
- hndl—Specifies the HTTP connection handle received for this request.
- returnCode—Specifies the error status for the HttpGetReply request. Zero indicates no errors; nonzero indicates an error. See Return Values for possible errors.
- httpStatusCode—Specifies the HTTP reply status, parsed from the first line of the HTTP reply headers. Valid only if the returnCode parameter is zero. For a list of possible values, see "HTTP Status Codes" on page 125.
- reserved—Reserved for future use. Currently returns NULL.

Return Values

If successful, returns 0. For an asynchronous request, success means that the parameters are valid and the request has been sent. For a synchronous request, success means that the request has been sent and the reply has been received.

On failure, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The handle was invalid, the call was made out of sequence, or the call was made while a callback on the handle is still pending.
255	ERR_BAD_PARAMETER	An invalid parameter was passed to the function.
>10000	WSA...	WinSock errors.

Remarks

You can call the HttpGetReply function immediately after the HttpOpenRequest function if your request doesn't require additional HTTP header tags or data. If called under these conditions, the HttpGetReply function sends the request, receives the reply, and parses the reply header.

The HttpGetReply function can be used synchronously or asynchronously. In synchronous mode, the function blocks until the send is completed and the status is returned. To use this mode, set the pContext and pCallBackRoutine parameters to NULL.

In asynchronous mode, the initial status returns immediately. If it is zero, the request has been accepted and your callback function is called when the reply is received and parsed. If the initial status is nonzero, an error has occurred, the request is not sent, and your callback function is never called.

See Also

HttpOpenRequest (page 99)

HttpOpenRequest

Provides information for an HTTP request.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpOpenRequest (
   HINTERNET   hndl,
   char       *pVerb,
   char       *pObjectName,
   char       *pVersion,
   char       *pReferer,
   char      **ppAcceptTypes);
```

Parameters

hndl

(IN) Specifies the connection handle returned from HttpConnect.

pVerb

(IN) Points to an ASCIIZ string that specifies the verb of the request (for example, GET, PUT, or POST).

pObjectName

(IN) Points to an ASCIIZ string that specifies the path to the object on the server. This is the portion of the URL that follows the domain name or address.

pVersion

(IN) Points to an ASCIIZ string that specifies the HTTP version for the request. If set to NULL, the current default version ("HTTP/1.1") is used for the string.

pReferer

(IN) Points to an ASCIIZ string that is the URL for the HTTP Referer header. If the Referer header is not needed, set the parameter to NULL.

ppAcceptTypes

(IN) Points to a NULL-terminated array of ASCIIZ string pointers that specify the media types accepted in the response. Each media specified is added to the Accept header in the request.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
1	ERR_INSUFFICIENT_SPACE	Insufficient space in header buffer.

Decimal	Name	Description
255	ERR_BAD_PARAMETER	An invalid parameter was passed to the function.

Remarks

The HttpOpenRequest function follows an HttpConnect call and completes the opening of a request. If all your request information is added with this function, it can be followed by the HttpGetReply function, which then both sends the request and receives the reply.

If you need to add more request information, it is followed by an HttpAddRequestHeader... function to add header information and by an HttpSendRequest function to enable the adding of data.

See Also

HttpAddRequestHeaderTag (page 89)
HttpConnect (page 95)
HttpGetReply (page 97)
HttpSendRequest (page 105)

HttpQueryOption

Returns the status of the global or connection HTTP options.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpQueryOption (
   HINTERNET   hndl,
   UINT32      option,
   void       *pBuffer,
   UINT32     *pBufferLen);
```

Parameters

hndl

(IN) Specifies one of the following handles:

- For global settings, the handle returned by the HttpClientOpen function.
- For connection settings, the handle returned by the HttpConnect function.

option

(IN) Specifies the option to view. Currently, only one option is available: HTTPCLNT_OPTION_CONNECT_TIMEOUT (3).

pBuffer

(OUT) Points to the buffer to receive the option information.

pBufferLen

(IN/OUT) Points to the length of pBuffer. Before calling this function, set this parameter to the maximum length of pBuffer. On a successful return, this parameter is set to the actual length of pBuffer. If pBuffer is too small for the response, returns the length of buffer needed.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
119	ERR_BUFFER_TOO_SMALL	The pBuffer parameter is too small for the response data.
255	ERR_BAD_PARAMETER	An invalid parameter was passed to the function.

Remarks

You can view the option globally for all your HTTP connections or individually for one connection.

See Also

HttpSetOption (page 107)

HttpReadResponseData

Returns HTTP reply body data.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpReadResponseData (
   HINTERNET   hndl,
   char      **pReplyDataBuffer,
   UINT32     *pPostDataLen,
   void       *pContext,
   void      (*pCallBackRoutine)(
               void       *pContext,
               HINTERNET   hndl,
               int         returnCode,
               char      **pReplyDataBuffer,
               UINT32     *pReplyDataLen,
               void       *reserved)
);
```

Parameters

hndl

(IN) Specifies the connection handle returned from HttpConnect.

pReplyDataBuffer

(IN/OUT) Points to a pointer to where the block of data is received.

pPostDataLen

(IN/OUT) Points to the length of the block of data received.

pContext

(IN) Points to optional context data, which is passed to your callback function. For a synchronous request, set it to NULL. Currently only synchronous mode is implemented.

pCallBackRoutine

(IN) Points to the callback function that is called when an asynchronous request returns. For a synchronous request, set it to NULL. Currently only synchronous mode is implemented.

The callback function is passed the following parameters:

- pContext—Points to any optional context data that you passed in when you called HttpReadResponseData.
- hndl—Specifies the HTTP connection handle received for this request.
- returnCode—Specifies the error status for the HttpReadResponseData call. Zero indicates no errors; nonzero indicates an error.
- pReplyDataLen—Points to a pointer to where the block of data is received.
- pReplyDataLen—Points to the length of the block of data received.

- reserved—Reserved for future use. Currently returns NULL.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
121	ERR_NO_ITEMS_FOUND	Either there is no body data in this reply, or all the data has been retrieved.
168	ERR_ACCESS_DENIED	The reply header is not valid for this request.
255	ERR_BAD_PARAMETER	An invalid parameter was passed to the function.

Remarks

IMPORTANT: Currently this function only works in synchronous mode. Both the pContext and pCallBackRoutine parameters must be set to 0.

The HttpReadResponseData function returns the address and length of the data received in the body of the HTTP rely. If the reply is chunked, this function should be called once for each chunk of the reply until the ERR_NO_ITEMS_FOUND error code is returned.

See Also

HttpCloseConnection (page 94)
HttpGetReply (page 97)
HttpQueryInfo (page 119)

HttpSendRequest

Opens the connection to the destination and sends the request.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpSendRequest (
   HINTERNET   hndl,
   void       *pDataBuffer,
   UINT32      dataBufferLen,
   void       *pContext,
   void      (*pCallBackRoutine)(
              void       *pContext,
              HINTERNET   hndl,
              int         returnCode,
              void       *reserved)
   );
```

Parameters

hndl

(IN) Specifies the connection handle returned from HttpConnect.

pDataBuffer

(IN) Points to optional data for the body of the request.

dataBufferLen

(IN) Specifies the length of the data pointed to by the pDataBuffer parameter.

pContext

(IN) Points to optional context data, which is passed to your callback function. For a synchronous request, set it to NULL.

pCallBackRoutine

(IN) Points to the callback function that is called when an asynchronous request returns. For a synchronous request, set it to NULL.

The callback function is passed the following parameters:

- pContext—Points to any optional context data that you passed in when you called HttpSendRequest.
- hndl—Specifies the HTTP connection handle received for this request.
- returnCode—Specifies the error status for the HttpSendRequest call. Zero indicates no errors; nonzero indicates an error. See Return Values for possible errors.
- reserved—Reserved for future use. Currently returns NULL.

Return Values

If successful, returns 0. For an asynchronous request, success means that the parameters are valid and the request has been sent. For a synchronous request, success means that the request has been sent and the reply has been received.

On failure, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The handle is invalid, the call was made out of sequence, or the call was made when a callback on the handle is pending.
255	ERR_BAD_PARAMETER	An invalid parameter was passed to the function.
>10000	WSA...	WinSock errors.

Remarks

Once a request has been built, you use the HttpSendRequest function to open the connection at the destination and send the request. You can supply the data portion of the request, in full or in part, with this function.

The HttpSendRequest function can be used synchronously or asynchronously. In synchronous mode, the function blocks until the send is completed and the status is returned. To use this mode, set the pContext and pCallBackRoutine parameters to NULL.

In asynchronous mode, the initial status returns immediately. If it is zero, the request has been accepted and your callback function is called when the reply is received and parsed. If the initial status is nonzero, an error has occurred, the request is not sent, and your callback function is never called.

See Also

HttpOpenRequest (page 99)
HttpSendDataSprintf (page 123)
HttpWriteRequestData (page 108)

HttpSetOption

Sets either a global or a connection option.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpSetOption (
   HINTERNET   hndl,
   UINT32      option,
   void       *pBuffer,
   UINT32      bufferLen);
```

Parameters

hndl

(IN) Specifies one of the following handles:

- For global settings, the handle returned by the HttpClientOpen function.
- For connection settings, the handle returned by the HttpConnect function.

option

(IN) Specifies the option to set. Currently, only one option is available: HTTPCLNT_OPTION_CONNECT_TIMEOUT (3).

pBuffer

(IN) Points to the buffer to containing the set data for the option.

pBufferLen

(IN) Points to the length of pBuffer.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
0xFF	ERR_BAD_PARAMETER	An invalid parameter was passed to the function.

Remarks

You can set the option globally for all your HTTP connections or individually for one connection.

See Also

HttpQueryOption (page 101)

HttpWriteRequestData

Sends the data portion of an HTTP request.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpWriteRequestData (
   HINTERNET   hndl,
   void       *pDataBuffer,
   UINT32     *pDataBufferLen,
   void       *pContext,
   void      (*pCallBackRoutine)(
              void       *pContext,
              HINTERNET   hndl,
              int         returnCode,
              void       *reserved)
   );
```

Parameters

hndl

(IN) Specifies the connection handle returned from HttpConnect.

pDataBuffer

(IN) Points to the data for the body of the request.

pDataBufferLen

(IN) Points to the length of the data pointed to by the pDataBuffer parameter.

pContext

(IN) Points to optional context data, which is passed to your callback function. For a synchronous request, set it to NULL.

pCallBackRoutine

(IN) Points to the callback function that is called when an asynchronous request returns. For a synchronous request, set it to NULL.

The callback function is passed the following parameters:

- pContext—Points to any optional context data that you passed in when you called HttpWriteRequestData.
- hndl—Specifies the HTTP connection handle received for this request.
- returnCode—Specifies the error status for the HttpWriteRequestData call. Zero indicates no errors; nonzero indicates an error. See Return Values for possible error codes.
- reserved—Reserved for future use. Currently returns NULL.

Return Values

If successful, returns 0. For an asynchronous request, success means that the parameters are valid and the data has been sent. For a synchronous request, success means that the data has been sent and the reply has been received.

On failure, returns a nonzero error code.

Decimal	Name	Description
168	ERR_ACCESS_DENIED	The handle is invalid, the call was made out of sequence, or the call was made when a callback on the handle is pending.
255	ERR_BAD_PARAMETER	An invalid parameter was passed to the function.
>10000	WSA...	WinSock errors.

Remarks

The HttpWriteRequestData function can be called as many times as necessary to send all the data.

The function can be used synchronously or asynchronously. In synchronous mode, the function blocks until the send is completed and the status is returned. To use this mode, set the pContext and pCallBackRoutine parameters to NULL.

In asynchronous mode, the initial status returns immediately. If it is zero, the data has been sent and your callback function is called when the data has been written. If the initial status is nonzero, an error has occurred, the data is not sent, and your callback function is never called.

See Also

HttpGetReply (page 97)
HttpSendDataSprintf (page 123)
HttpSendRequest (page 105)

5 HTTP Server and Client Functions

Server and client applications can use the functions documented in this section. This section alphabetically lists these functions from the httpexp.h file and describes their purpose, syntax, parameters, and return values.

HttpConvertName

Converts all reserved characters into escape sequences, and writes the result into a destination string.

Syntax

```
#include <httpexp.h>

void HttpConvertName (
  char   *src,
  char   *dst,
  UINT32  srclen);
```

Parameters

src

(IN) Points to an ASCII string.

dst

(OUT) Points to where the converted (Escape characters) string is placed. The resultant string is NULL terminated.

srclen

(IN) Specifies the length of the source ASCII string.

Remarks

For example, the string “computing power” would be converted to “computing%20power”.

See Also

HttpUnConvertName (page 124)

HttpExtractInternetTimeFromString

Extracts the Internet Time value from the ASCIIZ date and time string.

Syntax

```
#include <httpexp.h>

UINT32 HttpExtractInternetTimeFromString (
  char  *string);
```

Parameters

string

(IN) Points to an ASCIIZ string to extract the Internet time from.

Return Values

Returns the Internet Time (GMT) in seconds.

HttpFindNameAndValue

Finds the specified name and returns its value.

Syntax

```
#include <httpexp.h>

int HttpFindNameAndValue (
  char  *pBuffer,
  char  *pzTokenName,
  char  *pzTokenValue,
  int   *pzTokenValueMaxLen);
```

Parameters

pBuffer

(IN) Points to a NULL terminated buffer to search for the specified name

pzTokenName

(IN) Points to the NULL terminated name to find in the specified buffer. The check is case sensitive.

pzTokenValue

(OUT) Points to the buffer that receives the token value.

pzTokenValueMaxLen

(IN/OUT) Points to an int that contains the maximum length of the pzTokenValue buffer. On return, the length specifies the length of the value found or the length of the buffer needed if the original pzTokenValue buffer is too small.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
119	ERR_BUFFER_TOO_SMALL	The pzTokenValue buffer is too small.
121	ERR_NO_ITEMS_FOUND	Name does not exist in buffer.
255	ERR_BAD_PARAMETER	Invalid input parameters.

HttpInternetTimeToString

Converts the Internet Time value into a date and time ASCIIZ string.

Syntax

```
#include <httpexp.h>

int HttpInternetTimeToString (
  UINT32   utcTime,
  void    *ins);
```

Parameters

utcTime

(IN) Specifies the Internet time value.

ins

(OUT) Points to the string into which the formatted Internet Time will be written.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

HttpLocalTimeToInternetTime

Converts a DOS date and time value into an Internet Time value.

Syntax

```
#include <httpexp.h>

UINT32 HttpLocalTimeToInternetTime (
  UINT32   DOSDateAndTime);
```

Parameters

DOSDateAndTime

(IN) Specifies the DOS date and time value to convert.

Return Values

Returns Internet Time.

HttpParseBuffer

Parses a given buffer for an index and returns the name and value pair.

Syntax

```
#include <httpexp.h>

int HttpParseBuffer (
  char   *pBuffer,
  int     index,
  char   *pzTokenName,
  int    *pzTokenNameMaxLen,
  char   *pzTokenValue,
  int    *pzTokenValueMaxLen,
  char   *pzBufferProcessedTo);
```

Parameters

pBuffer

(IN) Points to a NULL terminated buffer to parse.

index

(IN) Specifies the index value of the Name-Value pair to parse for.

pzTokenName

(OUT) Points to the NULL terminated name to find in the specified buffer. The check is case sensitive.

pzTokenNameMaxLen

(IN/OUT) Points to an int that contains the maximum length of the pzTokenName buffer. On return, the length is the length of the name found or the length of the buffer needed if the original pzTokenName buffer is too small.

pzTokenValue

(OUT) Points to the buffer that receives the token value.

pzTokenValueMaxLen

(IN/OUT) Points to an int that contains the maximum length of the pzTokenValue buffer. On return, the length is the length of the value found or the length of the buffer needed if the original pzTokenValue buffer is too small.

pzBufferProcessedTo

(OUT) Points to where the pointer to the first character after the Name Value Pair is placed.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
119	ERR_BUFFER_TOO_SMALL	The pzTokenValue buffer is too small.
121	ERR_NO_ITEMS_FOUND	Name does not exist in buffer
135	ERR_CREATE_FILE_INVALID_NAME	The pzTokenName buffer does not point to a valid buffer.
255	ERR_BAD_PARAMETER	Invalid input parameters.

Remarks

The data in the buffer is in the format: &Name=value.

HttpQueryInfo

Extracts information from the message header.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpQueryInfo (
   HINTERNET   hndl,
   UINT32      dwInfoLevel,
   void       *pBuffer,
   UINT32     *pBufferLength);
```

Parameters

hndl

(IN) Specifies the client connection handle or the HTTP stack handle.

dwInfoLevel

(IN) Specifies a combination of an attribute to retrieve and the flags that modify the request. See "Query Request Information" on page 127.

pBuffer

(IN/OUT) Points to the buffer that receives the information.

pBufferLength

(IN/OUT) Points to the length of the buffer. When the function returns, points to the length of the information written to the buffer. When the function returns a string, the following rules apply:

- If the function succeeds, the length returned is the length of the string, in characters, minus 1 for the NULL-terminating character.
- If the function fails and the error returned is ERR_BUFFER_TOO_SMALL, the length returned indicates the number of bytes you must allocate to receive the string.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Hex	Name	Description
119	ERR_BUFFER_TOO_SMALL	The supplied buffer is too small.
121	ERR_NO_ITEMS_FOUND	The information is not present in the header.
255	ERR_BAD_PARAMETER	An invalid parameter was passed to the function.

Remarks

The HttpQueryInfo function is used to extract different named attributes from the message header into a caller supplied buffer.

HttpReturnString

Returns a pointer to the requested string type.

Syntax

```
#include <httpexp.h>

char *HttpReturnString (
  UINT32    stringType);
```

Parameters

stringType

(IN) Specifies the pre-defined string type. See “Common Content Types” on page 130.

Return Values

Returns the address of ASCIIZ string for requested content type.

HttpReturnDataTxBuffer

Returns the starting address and size of the Tx buffer.

Syntax

```
#include <httpexp.h>

void *HttpReturnDataTxBuffer (
  HINTERNET    hndl,
  UINT32_PTR   lpszBufferSize);
```

Parameters

hndl

(IN) Specifies the client connection handle or the HTTP stack handle.

lpszBufferSize

(OUT) Points to where the maximum size of the Tx buffer is placed.

Return Values

Returns the address of the Tx buffer.

HttpSendDataSprintf

Transmits formatted data to the body of a request.

Syntax

```
#include <httpexp.h>
#include <httpclnt.h>

int HttpSendDataSprintf (
   HINTERNET   hrequest,
   void       *controlString,
   ...);
```

Parameters

hrequest

(IN) Specifies the client connection handle or the HTTP stack handle.

controlString

(IN) Points to the address of the control string to be formatted into the HTTP data section.

...

(IN) Points to an argument for a conversion specifier. The number of arguments is determined by the format string.

Return Values

If successful, returns 0. Otherwise, returns a nonzero error code.

Decimal	Name	Description
162	ERR_IO_LOCKED	The application must complete the header before data functions are used.
168	ERR_ACCESS_DENIED	The handle is invalid, the call was made out of sequence, or the call was made when a callback on the handle is pending.
255	ERR_BAD_PARAMETER	A parameter is invalid.
>10000	WSA...	WinSock errors.

Remarks

You use the HttpSendDataSprintf function just as you would a printf function, which has a control string and additional arguments as required by the control string. The size of the output string should be limited to less than 1024 characters per call, to prevent internal buffer overflows. This function can be called multiple times to send all the data.

The function can be used to send either client request data or server response data.

HttpUnConvertName

Converts all escaped values into the appropriate character.

Syntax

```
#include <httpexp.h>

void HttpUnConvertName (
  char     *src,
  char     *dst,
  UINT32    srclen);
```

Parameters

src

(IN) Points to an ASCII string from which to convert Escaped characters.

dst

(OUT) Points to where to place the resulting string. The resulting string is NULL terminated.

srclen

(IN) Specifies the length of the source string.

See Also

HttpConvertName (page 112)

6 Values

This section also contains the following reference material:

HTTP Status Codes

The HTTP status codes are divided into four categories:

- 1xx codes are informational. The request was received and processing continues.
- 2xx codes indicate success. The action was successfully received, understood, and accepted.
- 3xx codes indicate redirection. Further action must be taken in order to complete the request.
- 4xx codes indicate client errors. The request contains bad syntax or cannot be processed.
- 5xx codes indicate server errors. The server failed to fulfill an apparently valid request.

Decimal	Name
100	HTTP_STATUS_CONTINUE
101	HTTP_STATUS_SWITCH_PROTOCOLS
200	HTTP_STATUS_OK
201	HTTP_STATUS_CREATED
202	HTTP_STATUS_ACCEPTED
203	HTTP_STATUS_PARTIAL
204	HTTP_STATUS_NO_CONTENT
205	HTTP_STATUS_RESET_CONTENT
206	HTTP_STATUS_PARTIAL_CONTENT
207	HTTP_STATUS_MULTI_STATUS
300	HTTP_STATUS_AMBIGUOUS

Decimal	Name
301	HTTP_STATUS_MOVED
302	HTTP_STATUS_REDIRECT
303	HTTP_STATUS_REDIRECT_METHOD
304	HTTP_STATUS_NOT_MODIFIED
305	HTTP_STATUS_USE_PROXY
307	HTTP_STATUS_REDIRECT_KEEP_VERB
400	HTTP_STATUS_BAD_REQUEST
401	HTTP_STATUS_DENIED
402	HTTP_STATUS_PAYMENT_REQ
403	HTTP_STATUS_FORBIDDEN
404	HTTP_STATUS_NOT_FOUND
405	HTTP_STATUS_BAD_METHOD
406	HTTP_STATUS_NONE_ACCEPTABLE
407	HTTP_STATUS_PROXY_AUTH_REQ
408	HTTP_STATUS_REQUEST_TIMEOUT
409	HTTP_STATUS_CONFLICT
410	HTTP_STATUS_GONE
411	HTTP_STATUS_LENGTH_REQUIRED
412	HTTP_STATUS_PRECOND_FAILED
413	HTTP_STATUS_REQUEST_TOO_LARGE
414	HTTP_STATUS_URI_TOO_LONG
415	HTTP_STATUS_UNSUPPORTED_MEDIA
500	HTTP_STATUS_SERVER_ERROR
501	HTTP_STATUS_NOT_SUPPORTED
502	HTTP_STATUS_BAD_GATEWAY
503	HTTP_STATUS_SERVICE_UNAVAIL
504	HTTP_STATUS_GATEWAY_TIMEOUT
505	HTTP_STATUS_VERSION_NOT_SUP

Query Request Information

Decimal Value	Name	Description of Returned Information
0	HTTP_QUERY_MIME_VERSION	Receives the version of the MIME protocol that was used to construct the message.
1	HTTP_QUERY_CONTENT_TYPE	Receives the content type of the resource (such as text/html).
2	HTTP_QUERY_CONTENT_TRANSFER_ ENCODING	Receives the additional content coding that has been applied to the resource. *Not Supported*
3	HTTP_QUERY_CONTENT_ID	Retrieves the content identification. *Not Supported*
4	HTTP_QUERY_CONTENT_DESCRIPTION	Obsolete. Maintained for legacy application compatibility only. *Not Supported*
5	HTTP_QUERY_CONTENT_LENGTH	Retrieves the size of the resource, in bytes.
6	HTTP_QUERY_CONTENT_LANGUAGE	Retrieves the language that the content is in.
7	HTTP_QUERY_ALLOW	Receives the methods supported by the server.
8	HTTP_QUERY_PUBLIC	Receives methods available at this server.
9	HTTP_QUERY_DATE	Receives the date and time at which the message was originated.
10	HTTP_QUERY_EXPIRES	Receives the date and time after which the resource should be considered outdated.
11	HTTP_QUERY_LAST_MODIFIED	Receives the date and time at which the server believes the resource was last modified.
12	HTTP_QUERY_MESSAGE_ID	*Not Supported*
13	HTTP_QUERY_URI	Receives some or all of the Uniform Resource Identifiers (URIs) by which the Request-URI resource can be identified.
14	HTTP_QUERY_DERIVED_FROM	*Not Supported*
15	HTTP_QUERY_COST	*Not Supported*
16	HTTP_QUERY_LINK	Obsolete. Maintained for legacy application compatibility only.
17	HTTP_QUERY_PRAGMA	Receives the implementation-specific directives that might apply to any recipient along the request/response chain.
18	HTTP_QUERY_VERSION	Receives the last response code returned by the server.

Decimal Value	Name	Description of Returned Information
19	HTTP_QUERY_STATUS_CODE	*Not Supported*
20	HTTP_QUERY_STATUS_TEXT	*Not Supported*
21	HTTP_QUERY_RAW_HEADERS	*Not Supported*
22	HTTP_QUERY_RAW_HEADERS_CRLF	*Not Supported*
23	HTTP_QUERY_CONNECTION	Retrieves any options that are specified for a particular connection and must not be communicated by proxies over further connections.
24	HTTP_QUERY_ACCEPT	Retrieves the acceptable media types for the response.
25	HTTP_QUERY_ACCEPT_CHARSET	Retrieves the acceptable character sets for the response.
26	HTTP_QUERY_ACCEPT_ENCODING	Retrieves the acceptable content-coding values for the response.
27	HTTP_QUERY_ACCEPT_LANGUAGE	Retrieves the acceptable natural languages for the response.
28	HTTP_QUERY_AUTHORIZATION	Retrieves the authorization credentials used for a request.
29	HTTP_QUERY_CONTENT_ENCODING	Retrieves any additional content codings that have been applied to the entire resource.
30	HTTP_QUERY_FORWARDED	*Not Supported*
31	HTTP_QUERY_FROM	Retrieves the e-mail address for the human user who controls the requesting user agent if the From header is given.
32	HTTP_QUERY_IF_MODIFIED_SINCE	Retrieves the contents of the If-Modified-Since header.
33	HTTP_QUERY_LOCATION	Retrieves the absolute URI (Uniform Resource Identifier) used in a Location response-header.
34	HTTP_QUERY_ORIG_URI	*Not Supported*
35	HTTP_QUERY_REFERER	Receives the URI (Uniform Resource Identifier) of the resource where the requested URI was obtained.
36	HTTP_QUERY_RETRY_AFTER	Retrieves the amount of time the service is expected to be unavailable.
37	HTTP_QUERY_SERVER	Retrieves information about the software used by the origin server to handle the request.
38	HTTP_QUERY_TITLE	*Not Supported*

Decimal Value	Name	Description of Returned Information
39	HTTP_QUERY_USER_AGENT	Retrieves information about the user agent that made the request.
40	HTTP_QUERY_WWW_AUTHENTICATE	Retrieves the authentication scheme and realm returned by the server.
41	HTTP_QUERY_PROXY_AUTHENTICATE	Retrieves the authentication scheme and realm returned by the proxy.
42	HTTP_QUERY_ACCEPT_RANGES	Retrieves the types of range requests that are accepted for a resource.
43	HTTP_QUERY_SET_COOKIE	Receives the value of the cookie set for the request.
44	HTTP_QUERY_COOKIE	Retrieves any cookies associated with the request.
45	HTTP_QUERY_REQUEST_METHOD	Receives the verb that is being used in the request, typically GET or POST.
46	HTTP_QUERY_REFRESH	*Not Supported*
47	HTTP_QUERY_CONTENT_DISPOSITION	Obsolete. Maintained for legacy application compatibility only.
48	HTTP_QUERY_AGE	Retrieves the Age response-header field. (not normally found in HTTP requests).
49	HTTP_QUERY_CACHE_CONTROL	Retrieves the cache control directives.
50	HTTP_QUERY_CONTENT_BASE	*Not Supported*
51	HTTP_QUERY_CONTENT_LOCATION	Retrieves the resource location for the entity enclosed in the message.
52	HTTP_QUERY_CONTENT_MD5	Retrieves an MD5 digest of the entity-body for the purpose of providing an end-to-end Message Integrity Check (MIC) for the entity-body. For more information, refer to RFC 1864.
53	HTTP_QUERY_CONTENT_RANGE	Retrieves the location in the full entity-body where the partial entity-body should be inserted and the total size of the full entity-body.
54	HTTP_QUERY_ETAG	Retrieves the entity tag for the associated entity.
55	HTTP_QUERY_HOST	Retrieves the Internet host and port number of the resource being requested.
56	HTTP_QUERY_IF_MATCH	Retrieves the contents of the If-Match request-header field.
57	HTTP_QUERY_IF_NONE_MATCH	Retrieves the contents of the If-None-Match request-header field.

Decimal Value	Name	Description of Returned Information
58	HTTP_QUERY_IF_RANGE	Retrieves the contents of the If-Range request-header field. This header allows the client application to check if the entity related to a partial copy of the entity in the client application's cache has not been updated. If the entity has not been updated, send the parts that the client application is missing. If the entity has been updated, send the entire updated entity.
59	HTTP_QUERY_IF_UNMODIFIED_SINCE	Retrieves the contents of the If-Unmodified-Since request-header field.
60	HTTP_QUERY_MAX_FORWARDS	Retrieves the number of proxies or gateways that can forward the request to the next inbound server.
61	HTTP_QUERY_PROXY_AUTHORIZATION	Retrieves the header that is used to identify the user to a proxy that requires authentication. This header can only be retrieved before the request is sent to the server.
62	HTTP_QUERY_RANGE	Retrieves the byte range of an entity.
63	HTTP_QUERY_TRANSFER_ENCODING	Retrieves the type of transformation that has been applied to the message body so it can be safely transferred between the sender and recipient.
64	HTTP_QUERY_UPGRADE	Retrieves the additional communication protocols that are supported by the server.
65	HTTP_QUERY_VARY	Retrieves the header that indicates that the entity was selected from a number of available representations of the response using server-driven negotiation.
66	HTTP_QUERY_VIA	Retrieves the intermediate protocols and recipients between the user agent and the server on requests, and between the origin server and the client on responses.
67	HTTP_QUERY_WARNING	Retrieves additional information about the status of a response that might not be reflected by the response status code.

Common Content Types

Decimal Value	Name
0	HTTP_CONTENT_TYPE_OCTET_STREAM
1	HTTP_CONTENT_TYPE_JPEG

Decimal Value	Name
2	HTTP_CONTENT_TYPE_GIF
3	HTTP_CONTENT_TYPE_PNG
4	HTTP_CONTENT_TYPE_ZIP
5	HTTP_CONTENT_TYPE_HTML
6	HTTP_CONTENT_TYPE_WAV
7	HTTP_CONTENT_TYPE_AU
8	HTTP_CONTENT_TYPE_MIDI
9	HTTP_CONTENT_TYPE_MP3
10	HTTP_CONTENT_TYPE_WML
11	HTTP_CONTENT_TYPE_HTML_UTF8

Request Method Types

Value	Name	Description
0	HTTP_REQUEST_METHOD_GET	The HTTP request packet contains a get request.
1	HTTP_REQUEST_METHOD_HEAD	The HTTP request packet contains a head request.
2	HTTP_REQUEST_METHOD_POST	The HTTP request packet contains a post request.
3	HTTP_REQUEST_METHOD_PUT	The HTTP request packet contains a put request.
4	HTTP_REQUEST_METHOD_DELETE	The HTTP request packet contains a delete request.
5	HTTP_REQUEST_METHOD_OPTIONS	The HTTP request packet contains an options request.
6	HTTP_REQUEST_METHOD_COPY	The HTTP request packet contains a copy request. *Currently Not Supported (WebDav verb)*
7	HTTP_REQUEST_METHOD_MOVE	The HTTP request packet contains a move request. *Currently Not Supported (WebDav verb)*
8	HTTP_REQUEST_METHOD_MKCOL	The HTTP request packet contains a mkcol request. *Currently Not Supported (WebDav verb)*

Value	Name	Description
9	HTTP_REQUEST_METHOD_PROPFIND	The HTTP request packet contains a propfind request. *Currently Not Supported (WebDav verb)*
10	HTTP_REQUEST_METHOD_PROPPATCH	The HTTP request packet contains a proppatch request. *Currently Not Supported (WebDav verb)*
11	HTTP_REQUEST_METHOD_LOCK	The HTTP request packet contains a lock request. *Currently Not Supported (WebDav verb)*
12	HTTP_REQUEST_METHOD_UNLOCK	The HTTP request packet contains a unlock request. *Currently Not Supported (WebDav verb)*
13	HTTP_REQUEST_METHOD_MKREF	The HTTP request packet contains a mkref request. *Currently Not Supported (WebDav verb)*
14	HTTP_REQUEST_METHOD_ORDERPATCH	The HTTP request packet contains a orderpatch request. *Currently Not Supported (WebDav verb)*

Information Bits

Value	Name	Definition
0x00000001	INFO_SUPERVISOR_PRIVILEGES_BIT	Indicates that the hndl parameter has been authenticated with Supervisor privileges.
0x00000002	INFO_CONSOLE_OPERATOR_PRIVILEGES_BIT	Indicates that the hndl parameter has been authenticated with Console Operator privileges.
0x00000004	INFO_LOGGED_IN_BIT	Indicates that the hndl parameter has been logged in and authenticated.
0x10000000	CONTROL_NO_AUTHENTICATION_OPTION_ENABLED_BIT	Indicates that the HTTP stack has disabled Authentication (the /NA option was selected when HttpStk was loaded).
0x20000000	CONTROL_NO_LOGIN_OPTION_ENABLED_BIT	Indicates that the HTTP stack has disabled logging in (the /NL option was selected when HttpStk was loaded).
0x40000000	CONTROL_DEINITIALIZATION_BIT	Indicates that the callback method is being invoked for de-initialization. All other bits are ignored if this bit is set.

Value	Name	Definition
0x80000000	CONTROL_INITIALIZATION_BIT	Indicates that the callback method is being invoked for initialization. All other bits are ignored if this bit is set.

Health Monitor Error Codes

The Health Monitoring Functions (page 28) return 0 if successful and one of the following error codes on failure:

Value	Name
1	INVALID_MODULE_HANDLE
2	INVALID_HEALTH_MONITOR_HANDLE
3	INVALID_CATEGORY
4	INVALID_DESCRIPTION_STRING
5	INVALID_URL_STRING
6	INVALID_HELP_URL_STRING
7	INSUFFICIENT_MEMORY
8	INVALID_REGISTER_FLAGS
9	HEALTH_MONITOR_REPLACED
10	MP_LOCK_ERROR
11	THRESHOLD_ALREADY_SET
12	INVALID_CALLBACK_ADDRESS
13	VALUE_NOT_AVAILABLE
14	UNABLE_TO_CREATE_KEY

Categories

The following table contains a list of the health monitor categories.

Category ID Number	Category and Description
0	KERNEL_SERVICES ◆ CPU Utilization ◆ Allocated Server Processes ◆ Available Server Processes ◆ DS Thread Usage ◆ Work To Do Response
1	DEBUG_SERVICES ◆ Abend Services
2	CONNECTION_SERVICES ◆ Connection Availability
3	FILESYSTEM_SERVICES ◆ Directory Cache Buffers ◆ Cache Performance ◆ Available Disk Space ◆ Available Directory Entries
4	MEMORY_SERVICES ◆ Available Memory ◆ Virtual Memory Performance ◆ Swap File Information
5	DIRECTORY_SERVICES ◆ DS Status
6	LAN_SERVICES Packet Receive Buffers Available ECBs LAN Traffic
7	STORAGE_SERVICES Disk Traffic
8	PRINTING_SERVICES ◆ NDPS Printing Services
9	APPLICATION_SERVICES ◆ GroupWise Status
10	HARDWARE_SERVICES ◆ Used for hardware vendors to report the state of their hardware.

Category ID Number	Category and Description
11	MISC

7 Structures

This section alphabetically lists the structures used by the NetWare Remote Manager functions and describes their purpose, syntax, and fields:

TOCregistration

Contains information for a table of contents entry.

Syntax

```
#include <httpexp.h>

struct TOCregistration
{
   UINT32   TOCHeadingNumber;
   char     reserved[4];
   char     TOCHeadingName[64];
};
```

Fields

TOCHeadingNumber

Set this field to a value between 0 and 6 to add your entry to that heading.

0	Diagnose Server
1	Manage Server
2	Manage Applications
3	Manage Hardware
4	Manage eDirectory
5	Use Group Operations
6	Storage Services

reserved

Must be set to zero.

TOCHeadingName

Contains the heading string that appears in the TOC.

Remarks

The RegisterServiceMethodEx function uses this structure to add a new item to the NetWare Remote Manager TOC.

8 Revision History

The following table lists all changes made to the NetWare Remote Manager documentation:

October 8, 2003	Restructured the manual and added an HTTP client services section for the HTTP request interfaces.
June 2000	Moved to Early Access section of the NDK.
March 2000	Added “Health Monitoring Functions” on page 28 and Common Page Look APIs.
March 2000	Initial release as a Leading Edge component on the NDK.