13 Webseiten von anderen Programmen abrufen

Es gehört zu den wesentlichen Funktionen für das Programmieren von Webservices, dass man eine bestimmte URL durch ein Programm abfragen kann. Dazu muss die URL an den Server gesendet und der zurückkommende Datenstrom entsprechend im Programm abgefangen werden, ohne dass sich etwa ein Browserfenster öffnet.

13.1 Abfragen einer URI mit Visual Basic

COM-Objekte für HTTP-Dienste

Windows stellt mehrere COM-Objekte zur Verfügung, um auf HTTP-Dienste zuzugreifen. Die beiden bedeutendsten sind `WINHTTP`, das ausschließlich für die NT-Technologie zur Verfügung steht, und die HTTP-Engine des Internet Explorers, die sich in der COM-Bibliothek `shdocvw.dll` versteckt.

13.1.1 Das Windows-Webbrowser-Control

Im folgenden Beispiel werden wir auf das Browser-Control des Internet Explorers zurückgreifen. Dabei nutzen wir aus, dass der Internet Explorer beim Aufruf einer URI den HTTP-Response-Block zunächst ordentlich in einem Objekt ablegt und dann den Browser aufruft, um die Seite zu rendern und darzustellen. Genau dieses Objekt werden wir auch aufrufen und dann die Antwort des Services aus dem Container des Objekts abholen.

Das nachstehende Beispiel zeigt die Wiederverwendung des Webbrowser-Controls. Der Aufruf eines `HTTP GET` ist ganz einfach, hierzu wird lediglich die URL der `Navigate`-Methode übergeben. Der `HTTP POST` funktioniert genauso, allerdings müssen vorher die HTTP-Header-Daten sinnvoll gesetzt und die Daten des HTTP-Bodys gefüllt werden. Die Werte für den Header hängen von der Applikation ab, die auf dem Webserver die Anfrage entgegennimmt. Die Daten für den Body müssen als Byte-Array übergeben werden, das von der Funktion `StrConv` erledigt wird.

Sehr wichtig ist in diesem Zusammenhang, dass man explizit auf die Beendigung des asynchronen Requests warten muss. Dies machen wir hier in einer Schleife, es könnte aber auch in einem Eventhandler stattfinden, der auf das Ende der Abfrage reagiert. Der VB-Befehl `DoEvents` ist

wichtig bei Schleifen, deren Ende unbestimmt ist und die daher gegebenenfalls ewig laufen. Der Befehl gibt die Kontrolle an den Eventhandler von Windows zurück, um diesem die Möglichkeit zu geben, einen anstehenden Event auszuführen, zum Beispiel das Drücken der Abbruch-Taste Ctrl-C.

Listing 13.1 Beispiel zur Verwendung des Webbrowser-Controls

```
Public Page As WebBrowser

Public Sub do_post(URL As String, body As Variant)
       Dim PostData() As Byte 'a local array with
       dynamic length
       PostData = StrConv(body, vbFromUnicode)
       Page.navigate URL, , , PostData, HTTP_Header
       While Page.Busy: DoEvents: Wend
End Sub

Public Sub do_get(URL As String)
       Page.navigate URL
       While Page.Busy: DoEvents: Wend
End Sub
```

13.1.2 Anwendungsbeispiel für das Browser-Control

Die Schwierigkeit beim Absetzen von HTTP-Kommandos ist meist das Finden der richtigen Werte für den Header. Das nachstehende Listing zeigt ein funktionierendes Beispiel, das den Request korrekt aufbaut und das Ergebnis in einer lokalen Datei abspeichert. Dieses Beispiel ist typisch und eignet sich deshalb gut als Vorlage für eigene Entwicklungen.

Listing 13.2 Vollständiges Listing eines VBA-Programms, das das IE-Webbrowser-Control wiederverwendet

```
Class ZZReuseIE
Public Page As WebBrowser
Public Document As MSHTML.HTMLDocument

Public User_Agent As String
Public Content_Type As String
Public Content_Length As Integer
Public Accept As String
Public Proxy_Connection As String
```

```
Public Referer As String
Public Host As String
Public Pragma1 As String
Public Accept_language As String
Public Accept_Encoding As String
Public Extension As String

Public Sub do_post(URL As String, body As Variant)
' Data to send to the server during the HTTP POST
' transaction.
' For example, the POST transaction is
' used to send data gathered by an HTML form.
' If this parameter does not specify any post data, the
' Navigate method issues an HTTP GET transaction.
' This parameter is ignored if URL is not an HTTP URL.
' NOTE: The post data specified by PostData is passed
' as a SAFEARRAY structure. The variant should
' be of type VT_ARRAY and point to a SAFEARRAY.
' The SAFEARRAY should be of element type
' VT_UI1, dimension one, and have an element count
' equal to the
' number of bytes of post data.
Dim PostData() As Byte 'a local array with dynamic
length
body = body + vbCrLf
PostData = StrConv(body, vbFromUnicode)
' Content_Length = Len(PostData) - 1
Page.navigate URL, , , PostData, HTTP_Header
' Page.navigate URL, , , body
While Page.Busy: DoEvents: Wend
Set Document = Page.Document
End Sub

Public Sub do_get(URL As String)
Page.navigate URL
While Page.Busy: DoEvents: Wend
Set Document = Page.Document
' do_post URL, ""
End Sub
```

```
Public Sub do_save(logtxt As String, Optional filename
As String)
Dim fs As Scripting.FileSystemObject
Dim logfile As Scripting.TextStream
If filename = "" Then filename = "U:\asplog.txt"
Set fs = CreateObject("Scripting.FileSystemObject")
Set logfile = fs.CreateTextFile(filename, True)
logfile.Write logtxt
logfile.Close
Set fs = Nothing
End Sub

Public Property Get HTTP_Header() As String
Dim Headers As String
Headers = ""
' HTTP_Header = "Content-Length: " +
' CStr(Content_Length) + vbCrLf
Add_Header_Line Headers, "Content-Type", Content_Type
Add_Header_Line Headers, "Extension", Extension
Add_Header_Line Headers, "User-Agent", User - Agent
Add_Header_Line Headers, "Host", Host
Add_Header_Line Headers, "Pragma", Pragma1
Add_Header_Line Headers, "Accept", Accept
Add_Header_Line Headers, "Proxy -Connection", Proxy_Con-
nection
Add_Header_Line Headers, "Referer", Referer
Add_Header_Line Headers, "Accept -Language", Accept_lan-
guage
Add_Header_Line Headers, "Accept -Encoding", Accept_
Encoding
HTTP_Header = Headers
End Property

Private Sub Add_Header_Line _
         (Header As String, tag As String, value As
          String)
If value <> "" Then
Header = Header + tag + ": " + value + vbCrLf
End If
End Sub
```

```
Private Sub Class_Initialize()
' Set Page = New WebBrowser
Set Page = CreateObject("InternetExplorer.Application")
Page.Visible = False

Referer = "VBA_app"
Accept_language = "*"
' Content type carries the MEDIA TYPE as defined in the
' Multipurpose Internet Mail Extension (MIME) specs
' Others may be: text/plain; charset=US-ASCII
' text/XML
Content_Type = "application/x-www-form-urlencoded"
Pragma1 = "no -cache"
Extension = "Security/Remote-Passphrase"
End Sub

Private Sub Class_Terminate()
Set Document = Nothing
Set Page = Nothing
End Sub
```

Nachdem wir die Klasse definiert haben, müssen wir sie nur noch mit geeigneten Mitteln aufrufen. Hier rufen wir eine URL als `HTTP GET` auf und speichern das Resultat in einer lokalen Datei.

Listing 13.3 Ein Hauptprogramm zur Klasse in Listing 13.2

```
Dim Request As New ZZReuseIE
Dim strHeader As String
Dim strForm As String
Const BaseURL = "http://logosworld.com/samples/nusskna-
cker.htm"

Sub Main()
Set Request = New HTTP
Request.do_get BaseURL
Debug.Print Request.Document.body.innerHTML
Request.do_save Request.Document.body.innerHTML,"nuss-
knackers.html"
End Sub
```

13.1.3 Das WINHTTP-Control

Sehr ähnlich dem Internet-Explorer-Control, allerdings in den meisten Fällen etwas einfacher zu bedienen, ist die WINHTTP-Library. Im Übrigen favorisiert Microsoft die Verwendung von WINHTTP in neuen Applikationen. Im folgenden Beispiel ist das WINHTTP-Objekt ein Element des MSXML-Objekts. Da Sie in den meisten Fällen ohnehin den HTTP-Dienst zusammen mit einem XML-Parser verwenden, ist die Verwendung des MSXML4-Objekts ganz praktisch.

Der Microsoft XML-Parser *MSXML* hat die volle Funktionalität des WINHTTP bereits eingebaut. Damit ist es besonders einfach, Webdienste über HTTP aufzurufen.

Falls Sie mit WINHTTP über einen Proxy-Server auf das Internet zugreifen wollen, müssen Sie die Benutzung des Proxys konfigurieren. Hierfür stellt Microsoft ein Tool *Proxycfg* zur Verfügung, das die notwendigen Einstellungen in der Windows-Registry vornimmt. Sie können dieses Tool gegebenenfalls von Microsoft MSDN herunterladen. Aufgerufen wird das Tool mit dem Befehl `Proxycfg -d`.

Das nachstehende Beispiel erzeugt zunächst eine Instanz des MSXML-Objekts und setzt die entsprechenden Parameter. Als URL wird eine Abfrage nach einem Artikel bei Amazon.com abgesetzt. Anders als beim Webbrowser-Control muss hier nicht explizit geprüft werden, ob der HTTP-Request schon abgeschlossen ist, vielmehr kommt die Abfrage mit `True` oder `False` zurück.

Listing 13.4 Request für ein XML-Dokument mit MSXML

```
Function AmazonASINsearch(ASIN As String) As MSXML2.DOM-
Document
  Dim SelectedText As String
  Dim MSXML As MSXML2.DOMDocument
  Dim XMLURL As String
  Dim Loaded As Boolean
  Set MSXML = CreateObject("MSXML.DOMDocument")
  MSXML.Async = False
  MSXML.preserveWhiteSpace = False
  MSXML.validateOnParse = True
  MSXML.resolveExternals = False
  XMLURL = "http://xml.amazon.com/onca/xml2" + _
           "?t=logosworldcom" + _
```

```
            "&dev-t=D2H301234JJ987" + _
            "&page=1" + _
            "&f=xml" + _
            "&mode=books" + _
            "&type=heavy" + _
            "&AsinSearch=" + ASIN

  Loaded = MSXML.Load(XMLURL)

  If (Loaded) Then
'    ProcessResults MSXML
  Else
    MsgBox "The service is not available."
  End If

  Set AmazonASINsearch = MSXML
End Function
```

13.1.4 Anwendungsbeispiel für das WINHTTP-Control

Das nachstehende Beispiel liest eine XML-Datei *ISBNSOAP.XML* von Datei und sendet diese als `HTTP POST`-Request via `WINHTTP` zum Server von Amazon.com. Auch hier handelt es sich um ein typisches Beispiel, das sich gut als Vorlage für eigene Abwandlungen eignet.

Listing 13.5 Vollständiges Listing eines VBA-Programms, das WINHTTP verwendet

```
Class ZZWinHTTP
Public Page As MSXML2.XMLHTTP
Public Document As MSHTML.HTMLDocument
Public User_Agent As String
Public Content_Type As String
Public Content_Length As Integer
Public Accept As String
Public Proxy_Connection As String
Public Referer As String
Public Host As String
Public Pragma1 As String
Public Accept_language As String
Public Accept_Encoding As String
Public Extension As String
Public SOAPAction As String
```

```
Public Sub do_post(URL As String, body As Variant)
' Data to send to the server during the HTTP POST trans-
action. For example, the POST transaction is
' used to send data gathered by an HTML form. If this
parameter does not specify any post data, the
' Navigate method issues an HTTP GET transaction. This
parameter is ignored if URL is not an HTTP
' URL.
' NOTE: The post data specified by PostData is passed as
a SAFEARRAY structure. The variant should
' be of type VT_ARRAY and point to a SAFEARRAY. The SAF-
EARRAY should be of element type
' VT_UI1, dimension one, and have an element count equal
to the number of bytes of post data.
    Dim PostData() As Byte 'a local array with dynamic
length
    Dim Flags As Long
    Dim TargetFrame As String

    Flags = 0
    TargetFrame = ""

    Page.Open "POST", URL, False

    SOAPAction = "urn:PI/DevCentral/SoapService" +
    vbCrLf
    PostData = StrConv(body, vbFromUnicode)
    Content_Length = UBound(PostData) - 1
    Content_Type = "text/xml; charset=""utf-8"""
    Me.HTTP_Header

    Page.send body
'    Page.navigate URL, , , body
'    While Page.Busy: DoEvents: Wend
    Do While Page.readyState < READYSTATE_INTERACTIVE:
    DoEvents: Loop
End Sub
```

```
Public Sub do_save(logtxt As String, Optional filename
As String)
    Dim fs As Scripting.FileSystemObject
    Dim logfile As Scripting.TextStream
    If filename = "" Then filename = "asplog.txt"
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set logfile = fs.CreateTextFile(filename, True)
    logfile.Write logtxt
    logfile.Close
    Set fs = Nothing
End Sub

Public Sub HTTP_Header()
'    HTTP_Header = "Content-Length: " +
     CStr(Content_Length) + vbCrLf
    Add_Header_Line "Content-Type", Content_Type
    Add_Header_Line "Extension", Extension
    Add_Header_Line "User-Agent", User - Agent
    Add_Header_Line "Host", Host
    Add_Header_Line "Pragma", Pragma1
    Add_Header_Line "Accept", Accept
    Add_Header_Line "Proxy -Connection",
    Proxy_Connection
    Add_Header_Line "Referer", Referer
    Add_Header_Line "Accept -Language", Accept_language
    Add_Header_Line "Accept -Encoding", Accept_Encoding
    Add_Header_Line "SOAPAction", SOAPAction
End Sub

Private Sub Add_Header_Line(tag As String, value As
String)
    If value <> "" Then
        Page.setRequestHeader tag, value
    End If
End Sub

Private Sub Class_Initialize()
'    Set Page = New WebBrowser
'    Set Page =
     CreateObject("InternetExplorer.Application")
```

```
    Set Page = CreateObject("MSXML2.XMLHTTP")
    Referer = "VBA_app"
    Accept_language = "*"
   Content_Type = "application/x-www-form-urlencoded"
' Content type carries the MEDIA TYPE as defined in the
' Multipurpose Internet Mail Extension (MIME) specs
' Others may be: text/plain; charset=US-ASCII
'                text/XML
    Pragma1 = "no -cache"
    Extension = "Security/Remote-Passphrase"
End Sub

Private Sub Class_Terminate()
    Set Document = Nothing
    Set Page = Nothing
End Sub
```

Jetzt fehlt nur noch ein Hauptprogramm, das unsere Klasse geeignet testet. Unser Beispiel liest nun eine vorbereitete SOAP-Abfrage aus einer Datei und sendet diese als `HTTP POST` an Amazon.com.

Listing 13.6 Ein Hauptprogramm zur Klasse ZZwinHTTP

```
Sub Main()
    Dim Request As New ZZWinHTTP
    Dim strHeader As String
    Dim strForm As String
    Dim myXML As New MSXML2.DOMDocument
    Request.do_save "Hello World"
    myXML.Load "ISBNSOAP.xml"
    Request.do_post BaseURL, myXML.XML
    Debug.Print Request.Page.responseText
    Request.do_save Request.Page.responseXML.XML, "ama-
zonresponse.xml"
End Sub
```

Zum besseren Verständnis hier noch einmal ein Beispiel für einen geeigneten SOAP-Request zur Abfrage der ISBN.

Listing 13.7 SOAP-Request zur Abfrage einer ISBN im Amazon-Katalog

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
```

```
SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC=
"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:si="http://soapinterop.org/xsd"
xmlns:typens="urn:PI/DevCentral1/SoapService">
    <SOAP-ENV:Body>
        <typens:AsinSearchRequest>
            <AsinSearchRequest
            xsi:type="typens:AsinRequest">
                <asin xsi:type="xsd:string">
                3528057297</asin>
                <tag xsi:type="xsd:string">
                logosworldcom</tag>
                <type xsi:type="xsd:string">
                lite</type>
                <dev-tag xsi:type="xsd:string">
                D2H3Y00012KJJ615</dev-tag>
            </AsinSearchRequest>
        </typens:AsinSearchRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

13.2 Webentwicklung mit SAP R/3

Webentwicklung ist ab R/3 3.0E möglich

Ab Release 6.10 hat die SAP-Basis eine zusätzliche Personality als Webserver bekommen. Dadurch verhält sich das SAP-Basissystem innerhalb eines Netzwerks wie ein HTTP-Server, akzeptiert also Anfragen via HTTP und kann Anfragen in HTTP aussenden. Damit bietet SAP eine ganze Reihe von alternativen Methoden, Entwicklungen für und um das Internet zusammen mit SAP R/3 zu realisieren. Sie können die SAP-Basis als Webserver einsetzen, von einem Webserver via RFC auf SAP R/3 zugreifen und von SAP R/3 via RFC externe Dienste nutzen. Allerdings stehen auch den älteren SAP R/3-Versionen die Webdienste über RFC zur Verfügung.

Web AS ist deutlicher erweiterter SAP-Kernel

Um ein Missverständnis auszuräumen: Der SAP Web Application Server ist weiterhin der bewährte stabile Kernel von SAP R/3, mit all den Funktionalitäten, mit denen das System so erfolgreich wurde. SAP Web AS ist

demnach das SAP Basissystem, erweitert um die Funktionalität eines Webservers, dessen herausragende Bedeutung durch den Namen zum Ausdruck gebracht wird.

Die Kommunikation zwischen einem rufenden HTTP-Client und dem Wen AS erfolgt über *Business Server Pages*.

13.2.1 SAP Function HTTP_GET

HTTP ist auch ohne Web AS via RFC möglich

Das Abfragen eines Webservice aus einer R/3-Instanz heraus ist auch problemlos via RFC möglich, was besonders wichtig ist, wenn Sie noch eine ältere Version von SAP R/3 einsetzen. Dazu verwenden wir den Funktionsbaustein HTTP_GET, der über eine RFC-Destination die URI an einen Webserver sendet und die HTTP-Response entgegennimmt.

RFC-Destination SAPHTTPA muss korrekt definiert sein

Die verwendete RFC-Destination verweist dabei auf ein Utility `SAPHTTPA` beziehungsweise `SAPHTTP`, die als RFC-Server definiert sind und als HTTP-Proxy funktionieren. `SAPHTTPA` ist dabei ein UNIX-Utility, das gewöhnlich auf dem R/3-Applikationsserver läuft, während `SAPHTTP` die gleiche Funktion erfüllt, aber über SAP GUI auf der Workstation eines interaktiven Users ausgeführt wird.

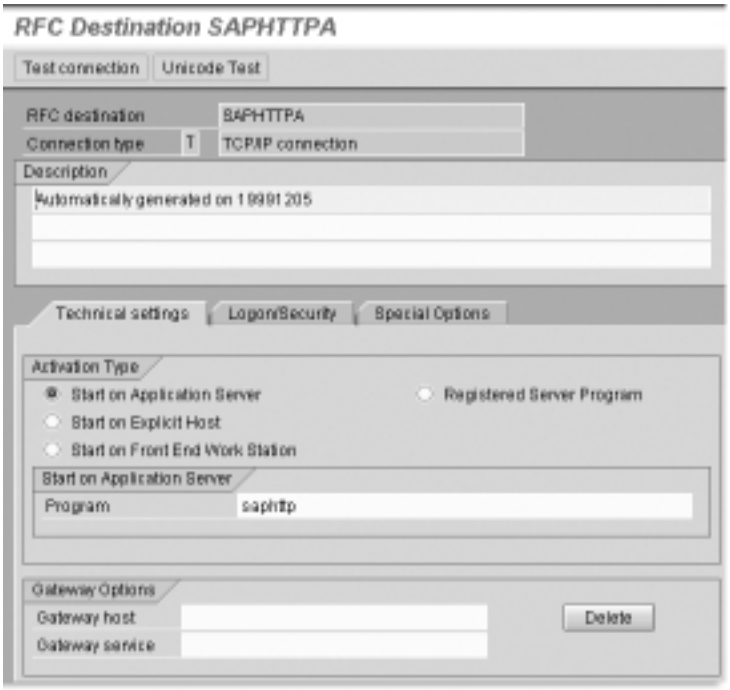

Abbildung 13.1 Definition der RFC-Destination SAPHTTPA in SM59 (Web AS)

Abbildung 13.2 Definition der RFC-Destination SAPHTTP mit Aufruf via Frontend (Release 4.6C)

Wenn wir sicher sind, dass die RFC-Destination `SAPHHTP` korrekt definiert ist, können wir darüber die URL von ABAP aufrufen. Dies ist mit jedem R/3-Release seit 3.1 möglich. Die URL zum Aufruf eines Amazon.com-Webservices kann wie folgt aussehen:

Listing 13.8 URI zum Finden einer ISBN (ASIN) bei Amazon.com

```
'http://xml.Amazon.com/onca/xml2?
t=webservices-20' &'&tag=logosworldcom
&dev-t=D2H3YO46KJJ615' &'&AsinS-
earch=3528057297&type=lite&f=xml'
```

Das nachstehende Beispiel führt den Aufruf durch und gibt das Ergebnis als ABAP-Liste aus.

Listing 13.9 Abruf einer URL mit ABAP

```
DATA: ABSOLUTE_URI(128) type c.

DATA: response_headers(80) occurs 0 with header line.
```

```
DATA: RESPONSE_ENTITY_BODY(120) occurs 0 with header
line.

ABSOLUTE_URI =
'http://xml.Amazon.com/onca/xml2?t=webservices-20' &
'&tag=logosworldcom&dev-t=D2H3YO46KJJ615' &
'&AsinSearch=3528057297&type=lite&f=xml'.

CALL FUNCTION 'HTTP_GET'
  EXPORTING
    ABSOLUTE_URI                     = ABSOLUTE_URI
    RFC_DESTINATION                  = 'SAPHTTPA'
    PROXY                            =
    '192.168.69.64:8080'
* IMPORTING
*   STATUS_CODE                      =
*   STATUS_TEXT                      =
*   RESPONSE_ENTITY_BODY_LENGTH      =
  TABLES
*   REQUEST_ENTITY_BODY              =
    RESPONSE_ENTITY_BODY             =
    RESPONSE_ENTITY_BODY
    RESPONSE_HEADERS                 =
    RESPONSE_HEADERS
*   REQUEST_HEADERS                  =
  EXCEPTIONS
    CONNECT_FAILED                   = 1
    TIMEOUT                          = 2
    INTERNAL_ERROR                   = 3
    TCPIP_ERROR                      = 4
    DATA_ERROR                       = 5
    SYSTEM_FAILURE                   = 6
    COMMUNICATION_FAILURE            = 7
    OTHERS                           = 8 .
IF SY-SUBRC <> 0.
  write: / sy-subrc.
ENDIF.

loop at response_entity_body.
  write: / response_entity_body.
endloop.
```

13.2.2 Proxy-Settings in ABAP

Falls ABAP die HTTP-Aufrufe über einen HTTP-Proxy ausführen muss, kann der Proxy für den ganzen Mandanten in der Tabelle THTTP vordefiniert werden. Dabei geht SAP davon aus, dass der erste Eintrag in THTTP die Angaben des Systemproxy darstellen.

Listing 13.10 Standardcoding zum Bestimmen des Proxy für ABAP HTTP-Requests

```
form set_HTTP_proxy using uri proxy proxy_user proxy_
password.
  data: proxyflag type c.
  if proxy ne space. exit. endif.
  select single * from tHTTP.
  if tHTTP-exitfunc ne space.
    call function tHTTP-exitfunc
         exporting
              absolute_uri = uri
         importing
              proxy        = proxyflag.
    if proxyflag eq 'X'.
      proxy = tHTTP-proxy.
      proxy_user = tHTTP-puser.
      proxy_password = tHTTP-ppassword.
    endif.
  endif.
endform.
```

Nachstehend sehen Sie zur besseren Orientierung die Struktur der Tabelle THTTP, in der die Proxy-Angaben hinterlegt sind. Meistens hat die Tabelle nur einen einzigen Eintrag.

MANDT	MANDT	CLNT	Client
PROXY	PROXY	CHAR	HTTP proxy host name
PUSER	PROXY_USER	CHAR	User name for HTTP proxy
PPASSWORD	PROXY_PWD	CHAR	Password for HTTP proxy
EXITFUNC	RS38L_FNAM	CHAR	Name of function module

Tabelle 13.1 Struktur der Tabelle THTTP

13.2.3 SAP und SOAP: Webservice Choreography Interface (WSCI)

SAP hat sich bisher noch nicht definitiv entschieden, WSDL vollständig zu unterstützen. Statt dessen ist SAP bemüht, zusammen mit BEA und SUN den neuen Interface-Standard WSCI populär zu machen. Zur Zeit der Drucklegung dieses Buches gibt es zwar eine vollständige Spezifikation von WSCI, aber noch keinerlei Marktakzeptanz, daher gehen wir hier nicht weiter darauf ein.

13.3 SAP Internet Transaction Server

Der *Internet Transaction Server* (ITS) war SAPs erster ernsthafter Versuch, die R/3-Technologie durch das Internet beziehungsweise Intranet zugänglich zu machen. Dabei handelt es sich um einen intelligenten Proxy-Server, der Daten aus den Dynpros einer Transaktionen extrahiert und sie in ein vordefiniertes HTML-Template einmischt.

Keine Programmiersequenzen

Im Grunde genommen funktioniert der ITS genauso wie ASP, JSP oder BSP. Allerdings kann man in den ITS-Seiten keine Programmiersequenzen einfügen, sondern nur Platzhalter für Variablen setzen, die später vom ITS ersetzt werden, bevor die so abgemischte Seite an den rufenden Browser weitergesandt wird.

Um das zu besser verstehen, rufen wir uns noch einmal in Erinnerung, wie R/3 mit dem SAP GUI kommuniziert. Ausgehend von einer bestehenden R/3-Sitzung, ist der Ablauf etwa wie folgt:

1. Das SAP GUI sendet die Daten eines Dynpros zu R/3
2. R/3 bestimmt das Folgedynpro und sendet
   - die Koordinaten der Dynproelemente zu SAP GUI
   - die Daten der Dynprofelder

ITS an Stelle des SAP GUI

Der ITS springt nun an Stelle des SAP GUI ein. Er filtert die Felddaten heraus und mischt sie in eine vordefinierte HTML-Seite ein. Diese HTML-Seiten sind im ITS-Repository abgespeichert und enthalten Platzhalter an den Stellen, wo R/3 Dynprodaten einfügen soll. Die so aufbereitete HTML-Seite wird dann durch den Internetbrowser aufgerufen.

Dieses Prinzip wurde in den Frühzeiten des World Wide Web von vielen Webservern angewandt, zum Beispiel auch von Microsofts Visual Studio. Dieses Abmischen von Daten mit Schablonen genügt heutigen Anforderungen bei weitem nicht mehr. Moderne Webserver übernehmen einen großen Teil der Aufbereitungslogik und lassen sich flexibel programmie-

ren, also mit IF-Bedingungen und Schleifen (LOOP, WHILE, FOR … NEXT). Somit ist der ITS bei weitem nicht mehr zeitgemäß. SAP hat deshalb die Weiterentwicklung von ITS schon seit Jahren aufgegeben und ersetzt ihn durch modernere Technologien, namentlich durch Business Server Pages und durch Web Dynpro.

Dank der RFC-Technologie kann man aber auch problemlos die Webseitengestaltung ganz aus SAP heraus verlagern. Im einfachsten Fall kann man auf ASP oder JSP zurückgreifen oder aber Messaging-Middleware für komplexe Sachverhalte zwischenschalten und die GUI-Funktionalität auf einen dedizierten Server – zum Beispiel CASABAC – auslagern.