

Webservices mit Konzept 16

von Florian Lapp

Das Web 2.0 bietet fast täglich neue Dienste mit wachsenden Datenmengen an. Mit Konzept 16 und seinen integrierten HTTP-, XML- und JSON-Funktionen lassen sich sowohl eigene als auch öffentliche Webservices wie Suchmaschinen und Handelsportale einfach ansteuern.

Jeder Webdienst unterscheidet sich in der Definition der Schnittstellen. Verschiedene Funktionen, Parameter und Rückgabewerte bestimmen die Kommunikation mit dem Service.

Die Integration erfolgt mit Konzept 16 aber auf einfache Art und Weise: Das HTTP-Objekt kümmert sich um die HTTP-Kommunikation. Die Interpretation und Aufbereitung der Daten übernehmen XML- und JSON-Funktionen. Der Entwickler muß lediglich die Schnittstellen des Webservices ansprechen. Über allgemeingültige Funktionen kann er verschiedene Schnittstellen unterschiedlicher Webservices einbinden.

Eine Schnittstelle für alle Schnittstellen

Mit der Funktion `WebSvcConnect()` wird eine TCP/IP-Verbindung zum Server hergestellt. Nach dem Aufbau der Verbindung kann die Applikation dann mehrere Anfragen hintereinander an den Webservice senden:

```
// ++++++
// + Verbindung herstellen +
// ++++++
sub WebSvcConnect(
    aHost      : alpha;      // Server-Name
    aPort      : word;      // Server-Port
)
: handle;      // Socket/Fehler

local
{
    tSck      : handle;
}

{
    tSck # SckConnect(aHost, aPort); // Verbindung herstellen

    return(tSck);
}
```

Nach der Verarbeitung der Anfragen wird die TCP/IP-Verbindung mit der Funktion `WebSvcDisconnect()` wieder getrennt.

```
// ++++++
// + Verbindung trennen +
// ++++++
```

```
sub WebSvcDisconnect(
    aSck      : handle;      // Socket
)
{
    aSck->SckClose(); // Verbindung trennen
}
```

`WebSvcSend()` sendet eine Anfrage an den Service. Im Parameter `aURIHost` wird der Host und in `aURIResource` die Ressource eines URI (Uniform Resource Identifier) übergeben. Der URI bildet die Schnittstelle zur Datenübertragung zum Webservice. In ihm sind die Funktion und alle dazugehörigen Parameter definiert.

```
// ++++++
// + Anfrage senden +
// ++++++
sub WebSvcSend(
    aSck      : handle;      // Socket
    aURIHost  : alpha;      // URI-Host
    aURIResource : alpha(4096); // URI-Ressource
)
: int;      // Fehler

local
{
    tErr      : int;
    tHTTP     : handle;
}

{
    // HTTP-Anfrage öffnen
    tHTTP # HTTPOpen(_HTTPSendRequest, aSck);
    if (tHTTP > 0)
    {
        tHTTP->spHostName # aURIHost; // Server-Name definieren
        tHTTP->spURI # aURIResource; // Ressource definieren
        tErr # tHTTP->HTTPClose(0); // HTTP-Anfrage senden
        // und schließen
    }
    else
    {
        tErr # tHTTP;
    }

    return(tErr);
}
```

Die Definitionen `_WebSvcFormatXML` und `_WebSvcFormatJSON` dienen zur Unterscheidung der Datenformate, denn es stehen zwei völlig unterschiedliche Verfahren zur Verfügung:

```
// +-----+
// + Datenformate +
// +-----+
define
{
    _WebSvcFormatXML : 0x00000001 // XML
    _WebSvcFormatJSON : 0x00000002 // JSON
}
```

Die Funktion `WebSvcRecv()` empfängt anschließend die Antwort des Dienstes. Im Parameter `aObj` kann entweder ein Speicherblock, eine Datei oder ein `CteNode`-Objekt übergeben werden, das die Schnittstelle zum Datenempfang vom Webservice bildet. Im Falle eines `CteNode`-Objektes läßt sich über `aOptions` mit den Optionen `_WebSvcFormatXML` und `_WebSvcFormatJSON` steuern, in welchem Format die Daten zu interpretieren sind. Der Parameter `vStatus` liefert den HTTP-Status der Antwort zurück.

```
// +-----+
// + Antwort empfangen +
```

```
// +-----+
sub WebSvcRecv(
    aSck : handle; // Socket
    aObj : handle; // Objekt
    var vStatus : int; // HTTP-Status
    opt aOptions : int; // Optionen
)
: int; // Fehler

local
{
    tObjType : int;
    tErr : int;
    tHTTP : handle;
    tObj : handle;
}
{
    vStatus # 0;
    tObjType # aObj->HdlInfo(_HdlType); // Objekttyp ermitteln
    // Objekt = Speicherblock oder Datei
    if (tObjType = _HdlMem or tObjType = _HdlFile)
    {
        tObj # aObj; // Objekt übernehmen
    }
    else
    {
        // Speicherblock allozieren
        tObj # MemAllocate(_MemAutoSize);
    }
}
```

```
01. {
02.   "ysearchresponse":{
03.     "responsecode":"200",
04.     "nextpage":"\ysearch/web/v1/%22CONCEPT%2016%22?format=json&count=1&appid=
05.     [...]&lang=de&region=de&start=1",
06.     "totalhits":"1259",
07.     "deephits":"8080",
08.     "count":"1",
09.     "start":"0",
10.     "resultset_web":[
11.       {
12.         "abstract":"Datenbank-Programmierung sowie Software-Entwicklung:
13.         ↳ <b>CONCEPT 16</b> ein Werkzeug zur effizienten Anwendungs-
14.         Programmierung<br>. <b>...</b> gemacht - Komponenten von <b>CONCEPT
15.         16</b> <b>...</b>",
16.         "clickurl":"http://lrd.yahooapis.com/[...]**http%3A
17.         ↳ \www.vectorsoft.de/",
18.         "date":"2009/08/12",
19.         "dispurl":"www.<b>vectorsoft.de</b>",
20.         "size":"22824",
21.         "title":"Datenbank Programmierung - Software Entwicklung",
22.         "url":"http://www.vectorsoft.de/"
23.       }
24.     ]
25.   }
26. }
```

Bild 1: Yahoo!-BOSS spricht JSON

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <ysearchresponse xmlns="http://www.inktomi.com/" responsecode="200">
03.   <nextpage><![CDATA[/ysearch/web/v1/%22CONCEPT%2016%22?format=json&count=1&appid=
04.   ↳ [...]&lang=de&region=de&start=1]]></nextpage>
05.   <resultset_web count="1" start="0" totalhits="1134" deephits="7910">
06.     <result>
07.       <abstract><![CDATA[Datenbank-Programmierung sowie Software-Entwicklung:
08.       ↳ <b>CONCEPT 16</b> ein Werkzeug zur effizienten Anwendungs-
09.       Programmierung<br>. <b>...</b> gemacht - Komponenten von <b>CONCEPT
10.       16</b> <b>...</b>]]></abstract>
11.       <clickurl>http://lrd.yahooapis.com/[...]**http%3A/www.vectorsoft.de
12.       ↳ /</clickurl>
13.       <date>2009/08/12</date>
14.       <dispurl><![CDATA[www.<b>vectorsoft.de</b>]]></dispurl>
15.       <size>22824</size>
16.       <title>Datenbank Programmierung - Software Entwicklung</title>
17.       <url>http://www.vectorsoft.de/</url>
18.     </result>
19.   </resultset_web>
20. </ysearchresponse>
```

Bild 2: Yahoo!-BOSS spricht XML

XML oder JSON?

Die überwiegende Mehrheit der Webservices basiert entweder auf XML und/oder auf JSON. XML ist eine mächtige Beschreibungssprache, JSON hingegen keine Sprache im eigentlichen Sinne. Ursprünglich wurde sie als Schreibweise für Objekte in JavaScript entworfen. Beide Formate haben ihre Vor- und Nachteile:

- JSON ist sehr viel kompakter als XML, das heißt die Daten können schneller transportiert werden, außerdem ist das Parsen von JSON-Daten weitaus weniger komplex als bei XML-Daten.
- XML bietet eine bessere Überprüfung (Validierung) der Daten als das bei JSON möglich ist: Die Daten können mit einem vorgegebenen Schema auf ihre Gültigkeit überprüft werden. Diese Technik findet bei der Kommunikation mit Webservices jedoch kaum Anwendung.

Datenbanken

```
// HTTP-Antwort empfangen
tHTTP # HTTPOpen(_HTTPRecvResponse, aSck);
if (tHTTP > 0)
{
    // HTTP-Status ermitteln
    vStatus # CnvIA(StrCut(tHTTP->spStatusCode, 1, 3));
    // HTTP-Daten ermitteln
    tErr # tHTTP->HTTPGetData(tObj);
    // HTTP-Antwort schließen
    tHTTP->HTTPClose(0);
}
else
{
    tErr # tHTTP;
}
// Objekt != Speicherblock oder Datei
if (tObjType != _HdlMem and tObjType != _HdlFile)
{
    if (tErr = _ErrOK)
    {
        // Unterscheidung über Objekttyp
        switch (tObjType)
        {
            case _HdlCteNode :
            {
                // Unterscheidung über Datenformat
                switch (aOptions & (_WebSvcFormatXML |
                    _WebSvcFormatJSON))
                {
                    // XML-Daten
                    case _WebSvcFormatXML :
                    {
                        // Daten als XML-Daten interpretieren
                        // und in CteNode-Objekt laden
                        if (aObj->XMLLoad('', 0, tObj) != _ErrOK)
                        {
                            tErr # _ErrData;
                        }
                    }
                    // JSON-Daten
                    case _WebSvcFormatJSON :
                    {
                        // Daten als JSON-Daten interpretieren
                        // und in CteNode-Objekt laden
                        if (aObj->JSONLoad('', 0, tObj) != _ErrOK)
                        {
                            tErr # _ErrData;
                        }
                    }
                }
            }
        }
    }
    tObj->MemFree(); // Speicherblock freigeben
}
return(tErr);
}
```

WebSvcURISplit() zerlegt einen im Parameter *aURI* übergebenen URI in seine Bestandteile. In *vURIHost* liefert sie den Server-Namen, in *vURIPort* den Server-Port und in *vURIResource* die Ressource zurück:

```
// ++++++
// + URI zerlegen
// ++++++
sub WebSvcURISplit(
    aURI          : alpha(4096); // URI
    var vURIHost  : alpha;       // URI-Host
    var vURIPort  : word;        // URI-Port
    var vURIResource : alpha;    // URI-Resource
)
: logic; // Erfolg
local
{
    tLen      : int;
    tPosBegin : int;
    tPosEnd   : int;
}
{
    vURIHost # '';
    vURIPort # 0;
    vURIResource # '';
    tLen # StrLen(aURI);
    tPosBegin # StrFind(aURI, '///', 1);
    if (tPosBegin > 0)
    {
        inc (tPosBegin, 2);
        tPosEnd # StrFind(aURI, '/', tPosBegin);
        if (tPosEnd = 0)
        {
            tPosEnd # tLen + 1;
        }
        vURIHost # StrCut(aURI, tPosBegin, tPosEnd - tPosBegin);
        tPosBegin # StrFind(vURIHost, ':', 1);
        if (tPosBegin > 0)
        {
            vURIPort # CnvIA(StrCut(vURIHost, tPosBegin + 1, 80));
            vURIHost # StrCut(vURIHost, 1, tPosBegin - 1);
        }
        vURIResource # StrCut(aURI, tPosEnd, tLen - tPosEnd +
            1);
        return(true);
    }
    return(false);
}
}
```

Die Funktionen zur TCP/IP- und HTTP-Verarbeitung sind bei Konzept 16 komfortabel in *WebSvcProcess()* zu einer einzigen Funktion gekapselt. Sie sorgt für den Auf- und Abbau der Verbindung, den Versand einer Anfrage und den Empfang einer Antwort. Im Parameter *aURI* wird der URI (die Schnittstelle zur Datenübertragung) und in *aObj* ein Objekt (die Schnittstelle zum Datenempfang) übergeben. Über *vStatus* liefert die Funktion den HTTP-Status der Antwort zurück. Mit dem Parameter *aOptions* wird das Datenformat angegeben:

```
// ++++++
// + Verarbeitung
// ++++++
sub WebSvcProcess(
    aURI          : alpha(4096); // URI
    aObj          : handle;      // Objekt
```

Datenbanken

```

var vStatus      : int;      // HTTP-Status
opt aOptions     : int;      // Optionen
)
: int;
local
{
  tURIHost       : alpha;
  tURIPort       : word;
  tURIResource   : alpha(4096);
  tErr           : int;
  tSck           : handle;
}
{
  vStatus # 0;
  // URI zerlegen
  if (WebSvcURISplit(aURI, var tURIHost,
                    var tURIPort, var tURIResource))
  {
    if (tURIPort = 0) // Kein Port angegeben
    {
      tURIPort # 80; // Standardport 80 verwenden
    }
    // Verbindung herstellen
    tSck # WebSvcConnect(tURIHost, tURIPort);
    if (tSck > 0)
    {
      // Anfrage senden

```

```

tErr # tSck->WebSvcSend(tURIHost, tURIResource);
if (tErr = _ErrOK)
{
  // Antwort empfangen
  tErr # tSck->WebSvcRecv(aObj, var vStatus,
                        aOptions);
}
tSck->WebSvcDisconnect(); // Verbindung trennen
}
else
{
  tErr # tSck;
}
}
else
{
  tErr # _ErrGeneric;
}
return(tErr);
}

```

Viele große Internet-Plattformen bieten ihre Dienste mittlerweile nicht nur klassisch über den Browser, sondern auch über Webservices an wie beispielsweise Suchmaschinen, Handelsportale, Bilder- und Dateiverwaltungsdienste. Das folgende Beispiel der Suchmaschine Yahoo! zeigt, wie

Datenbanken

die Webservices mit Konzept 16 in Applikationen integriert werden. Yahoo! taufte seinen Suchdienst auf den Namen BOSS (Build your Own Search Service; englisch für »Bau deinen eigenen Suchdienst«). Eine ausführliche Dokumentation zu BOSS ist unter [1] zu finden. Um den Dienst zu anzusprechen, benötigt man eine »Application ID«, eine Art Authentifizierungsschlüssel, den man nach einer Registrierung unter [2] anfordert. Die Applikation muß diese ID bei jeder Anfrage an den Dienst mit angeben. Die Funktion `WebSvcProcessYahooSearchWeb()` führt eine Web-Suche bei Yahoo! durch. Im Parameter `aQuery` wird die Suchanfrage übergeben. Die Application ID wird in `aAppID` mitgeführt. Die Argumente `aLang` und `aRegion` definieren die Sprache und die Region der Suchergebnisse.

```
sub WebSvcProcessYahooSearchWeb(
    aQuery      : alpha(1024); // Anfrage
    aAppID     : alpha;       // Application ID
    opt aLang  : alpha( 2); // Sprache
    opt aRegion : alpha( 2); // Region
)
local
{
    tCteNode      : handle;
    tErr          : int;
    tStatus      : int;

    tCteNodeResultList : handle;
    tCteNodeResultItem : handle;
    tCteNodeResultInfo : handle;

    tTitle       : alpha(4096);
    tURL         : alpha(4096);
}
{
    // CteNode-Objekt erzeugen
    tCteNode # CteOpen(_CteNode, _CteChildList |
                     _CteChildTreeCI);

    // Suche durchführen
    tErr # WebSvcProcess(
        'http://boss.yahooapis.com' // Yahoo-Boss-Server
        + '/ysearch/web/v1/'
        + StrCnv(aQuery, _StrToURI) // Anfrage
        + '?appid=' + aAppID       // Application ID
        + '&lang=' + aLang         // Sprache
        + '&region=' + aRegion     // Region
        + '&format=json'         // Format
        , tCteNode
        , var tStatus             // HTTP-Status
        , _WebSvcFormatJSON      // Format
    );
    // Suche erfolgreich
    if (tErr = _ErrOK and tStatus = 200)
    {
        // Nach Ergebnisliste suchen
        tCteNodeResultList # tCteNode->CteRead(
            _CteNodePath, 0, 'ysearchresponse/resultset_web');
        // Ergebnisliste gefunden
        if (tCteNodeResultList > 0)
        {
            // Ergebnisse iterieren
            for tCteNodeResultItem # tCteNodeResultList->CteRead(
                _CteChildList | _CteFirst);
        }
    }
}
```

```
loop tCteNodeResultItem # tCteNodeResultList->CteRead(
    _CteChildList | _CteNext, tCteNodeResultItem);
while (tCteNodeResultItem > 0)
{
    tTitle # '';
    tURL # '';
    // Titel suchen
    tCteNodeResultInfo # tCteNodeResultItem->CteRead(
        _CteNodePath, 0, 'title');
    if (tCteNodeResultInfo > 0)
    {
        // Titel ermitteln
        tTitle # StrCnv(tCteNodeResultInfo->spValueAlpha,
            _StrFromUTF8
        );
    }
    // URL suchen
    tCteNodeResultInfo # tCteNodeResultItem->CteRead(
        _CteNodePath, 0, 'url');
    if (tCteNodeResultInfo > 0)
    {
        // URL ermitteln
        tURL # StrCnv(tCteNodeResultInfo->spValueAlpha,
            _StrFromUTF8
        );
    }
    // ...
    // weitere Verarbeitung
    // ...
}
}
tCteNode->CteClear(true); // CteNode-Objekt leeren
tCteNode->CteClose();    // CteNode-Objekt schließen
}
```

Die Funktion führt eine Suche durch und verarbeitet die Ergebnisse anschließend in einer Schleife. Mit dem folgenden Aufruf und einer gültigen Application ID wird nach dem Begriff »CONZEPT 16« in den Ergebnissen aus Deutschland gesucht:

```
WebSvcProcessYahooSearchWeb('CONZEPT 16',
    '{appid}',
    'de', 'de');
```

Neben der Web-Suche bietet Yahoo! weitere Optionen an, wie beispielsweise eine Bilder- oder eine News-Suche. Die Webservices anderer Anbieter wie Google, Amazon oder eBay werden auf ähnliche Weise in Konzept-16-Applikationen integriert und werden so ein Teil der Vernetzung des Web 2.0.

Links

- [1] Yahoo! Developer Network: BOSS API Guide: http://developer.yahoo.com/search/boss/boss_guide/
- [2] Yahoo! Developer Network: Developer Registration: <http://developer.yahoo.com/wsregapp/>