HTTP

KNIHOVNA PRO IMPLEMENTACI HTTP SERVERU

Příručka uživatele a programátora

***SofCon*®spol. s r.o.**
Střešovická 49
162 00 Praha 6
tel/fax: +420 220 180 454
E-mail: sofcon@sofcon.cz
www: http://www.sofcon.cz

Verze dokumentu 1.10

Informace v tomto dokumentu byly pečlivě zkontrolovány a SofCon věří, že jsou spolehlivé, přesto SofCon nenese odpovědnost za případné nepřesnosti nebo nesprávnosti zde uvedených informací.
SofCon negarantuje bezchybnost tohoto dokumentu ani programového vybavení, které je v tomto dokumentu popsané. Uživatel přebírá informace z tohoto dokumentu a odpovídající programové vybavení ve stavu, jak byly vytvořeny a sám je povinen provést validaci bezchybnosti produktu, který s použitím zde popsaného programového vybavení vytvořil.
SofCon si vyhrazuje právo změny obsahu tohoto dokumentu bez předchozího oznámení a nenese žádnou odpovědnost za důsledky, které z toho mohou vyplynout pro uživatele.

Datum vydání: 16.05.2003
Datum posledního uložení dokumentu: 16.05.2003
(Datum vydání a posledního uložení dokumentu musí být stejné)

Obsah :

1. O dokumentu

1.1. Revize dokumentu

Verze dokumentu	Verze SW	Autor	Datum vydání	Popis změn
1.00	1.XX	Čr		První vydání
1.10	1.XX	Tu	16.05.2003	Úprava dokumentu dle ISO9000

1.2. Účel dokumentu

Tento dokument slouží jako popis jednotky implementující HTTP server.

1.3. Rozsah platnosti

Určen pro programátory a uživatele programového vybavení SofCon.

1.4. Související dokumenty

Pro čtení tohoto dokumentu je potřeba seznámit se s manuálem CoBase, CoInet a CoTcp.

Popis formátu verze knihovny a souvisejících funkcí je popsán v manuálu LibVer.

2. Termíny a definice

Používané termíny a definice jsou popsány v samostatném dokumentu Termíny a definice.

3. Úvod

3.1. Účel knihovny HTTP

Informační technologie jsou stále více aplikovány do celé řady přístrojů vzájemně propojených do rozličných sítí pomocí standardních síťových protokolů. Sítě lokálního a globálního charakteru se dnes již staly nedílnou součástí moderní společnosti. Takovéto sítě mohou tvořit nejen osobní počítače, ale i rozličné mikroprocesorové systémy. Spojením celé řady zařízení a senzorů, vzniká zcela nový komplexní celek, umožňující sběr, sdílení a zpracování dat. Jednou z moderních a komfortních metod použitelnou pro komunikaci mezi takovými systémy je architektura klient-server s protokolem HTTP.

Tato knihovna implementuje HTTP protokol podle standardu RFC-1945 (HTTP 1.0) a umožňuje velice jednoduchým způsobem doplnit i již existující aplikace o plnohodnotný HTTP server. Použití knihovny HTTP vyžaduje pouze základní znalosti protokolu HTTP.

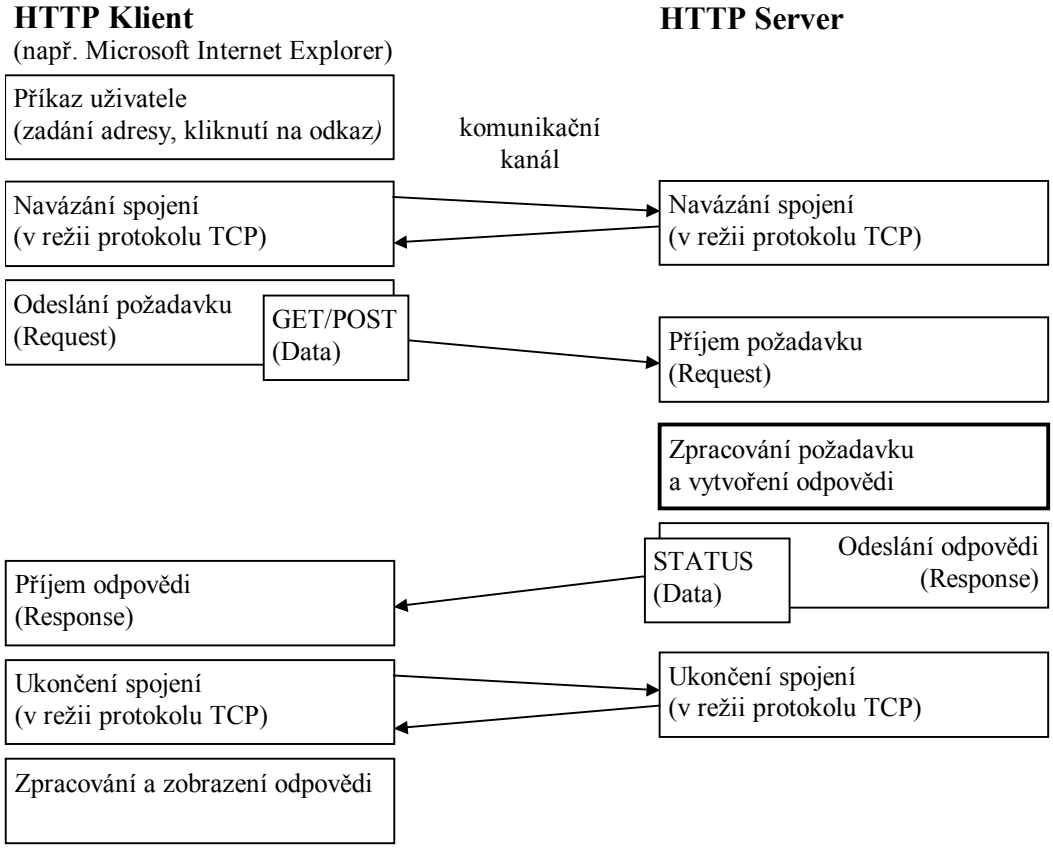
Tento dokument se na mnoha místech odkazuje na standardy RFC (Request for Comments), konkrétně na specifikace protokolu HTTP 1.0 v RFC-1945 a HTTP 1.1 v RFC-2068. Tyto standardy lze získat např. z adresy www.rfc.org

Pozn.: Pokud přidáte podporu http protokolu do aplikace, která do té doby nevyužívala ethernetovou kartu a protokoly TCP/IP, počítejte s nárůstem kódu v paměti FLASH o cca 100KB.

3.2. Model komunikace pomocí protokolu HTTP

Komunikace pomocí protokolu HTTP funguje na bázi klient-server. Server je zařízení příp. aplikace poskytující služby a data tzv. klientům. HTTP server a klient komunikují pomocí HTTP protokolu. HTTP klient navazuje spojení s HTTP serverem za účelem odeslání požadavku a příjmu odpovědi. Klientem může být Internetový prohlížeč (např. Microsoft Internet Explorer) nebo i jednoúčelová aplikace komunikující pomocí se serverem pomocí protokolu HTTP. HTTP server na základě požadavků klientů může provádět různé akce a vracet požadovaná data.

Základní komunikační jednotkou je **relace** (Session). Relaci iniciuje **klient** (User-Agent) pokusem o navázání spojení se serverem. Vyšle **požadavek** (Request), který je serverem zpracován. Po doručení **odpovědi** (Response) serveru je spojení uzavřeno a relace ukončena. Podrobněji jednu relaci popisuje následující obrázek:



HTTP protokol využívá k přenosu vlastních dat transportní protokol TCP. Tento protokol zajišťuje spolehlivý přenos s navazováním spojení (viz. manuál ke knihovně CoTCP). Detailní popis HTTP protokolu je uveden ve standardech RFC-1945 (verze 1.0) a RFC-2068 (verze 1.1). Knihovna http implementuje protokol verze 1.0, avšak některé vlastnosti (jako např. chybové kódy) jsou převzaty z verze 1.1.

3.3. Protokol HTTP

Protokol HTTP (Hypertex Transfer Protocol) je aplikační protokol architektury TCP/IP.. HTTP protokol je postaven nad spolehlivým potvrzovaným protokolem transportní vrstvy TCP (Transmission Control Protocol). HTTP je obecně bezestavový a objektově orientovaný protokol, který může být použit k mnoha účelům. V obvyklých případech se pomocí protokolu HTTP přenášejí data ve formátu HTML (Hypertext Markup Language), nicméně tento protokol je natolik variabilní, že umožňuje předávat data prakticky libovolná, např. binární, ve formátu XML (Extensible Markup Language) apod.

3.3.1. HTTP požadavek

Příklady požadavků:

`GET /index.html?par=1 HTTP1/0`
`User-Agent: Mozzila 4/0 (compatible)`
`Accept: */*`
	Požadavek na získání dokumentu /index.html s parametrem “par” s

	hodnotou “1”.
POST /proc/data HTTP1/0 User-Agent: Mozzila 4/0 (compatible) Accept: */* Content-Type: text/plain Content-Length: 10 Content: 1234567890	Požadavek odesílající textová data “1234567890” na server na absolutní cestu /proc/data.

Každý požadavek se skládá z několika částí. Povinný je pouze první řádek (příkazový). Ostatní části jsou nepovinné a jsou klientem do požadavku zahrnuty podle potřeby. Požadavek se skládá z:

- příkazový řádek obsahující příkaz, název dokumentu, parametry a verzi protokolu
- obecné záhlaví obsahující položky společné jak pro požadavek, tak pro odpověď (např. Date, Pragma)
- záhlaví požadavku obsahující položky specifické pro požadavek (např. If-Modified-Since, Referer, User-Agent, Accept)
- záhlaví entity obsahující položky specifické pro požadavek nesoucí data (entitu) (např. Allow, Content-Type, Content-Encoding, Content-Length, Last-Modified)
- data entity

3.3.2. HTTP odpověd

Příklady odpovědí:

HTTP1/0 200 OK Server: KIT386 Content-Type: text/html Content-Length: Content: 245 <html> <body> . . . </body> </html>	
HTTP1/0 404 Not Found Server: KIT386	

Každá odpověď serveru se skládá z několika částí. Povinný je pouze první řádek (stavový). Ostatní části jsou nepovinné a jsou do odpovědi serveru zahrnuty podle potřeby. Odpověď se skládá z:

- stavový řádek obsahující verzi protokolu, chybový kód a text upřesňující chybový kód
- obecné záhlaví obsahující položky společné jak pro požadavek, tak pro odpověď (např. Date, Pragma)
- záhlaví odpovědi obsahující položky specifické pro odpověď (např. Location,

Server)
- záhlaví entity obsahující položky specifické pro požadavek nesoucí data (entitu) (např. Allow, Content-Type, Content-Encoding, Content-Length, Last-Modified)
- data entity

3.3.3. Příkazový řádek požadavku

Příkazový řádek je nejdůležitější částí požadavku. Vždy začíná identifikátorem metody, následuje požadovaný URI (Unified Resource Identifier), tj. cesta k dokumentu. Poslední položkou příkazového řádku je verze protokolu. Příklad:

```
GET /pub/WWW/TheProject.html HTTP/1.0
```

URI může označovat dokument na serveru nebo např. identifikátor procesu, který se spustí a provede nějakou akci atd.

Typ metody určuje jakým způsobem se má naložit s požadovaným URI. Nejčastěji používané metody jsou GET a POST.

- Pomocí metody **GET** jsou žádána data ze serveru (požadavek obsahuje pouze příkazový řádek a případně obecné záhlaví a záhlaví požadavku, neobsahuje však data entity).
- Pomocí metody **POST** jsou na server zasílána data. Požadavek navíc obsahuje záhlaví entity a vlastní data.

Součástí URI mohou být také další parametry, které jsou od cesty k dokumentu odděleny znakem '?'. Např. dotaz:

```
GET /parametry.xml?zarizeni=5&sada=2 HTTP/1.0
```

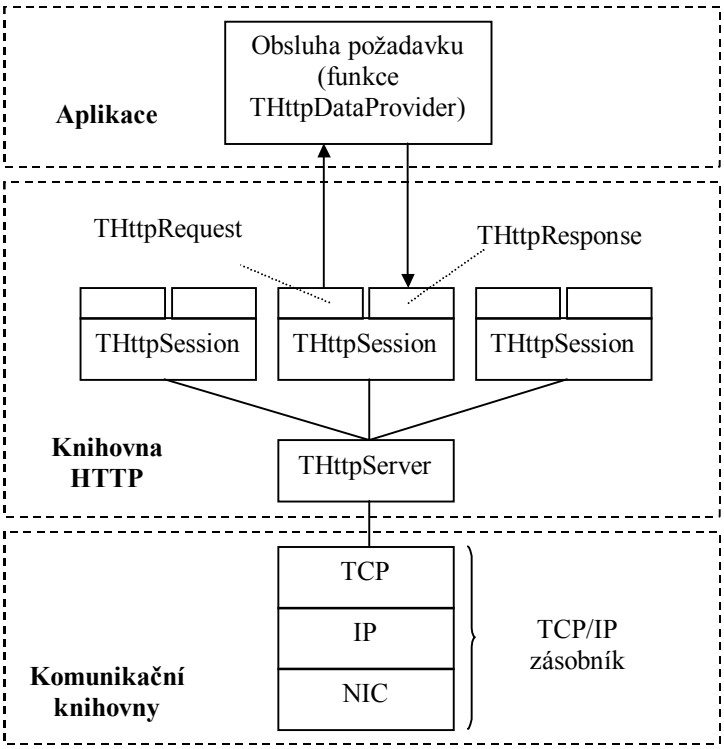
obsahuje parametry "zarizeni" a "sada".

3.4. Struktura knihovny HTTP

Knihovna HTTP definuje několik nových tříd. Nejdůležitější z nich jsou:

THttpServer	Zajišťuje inicializaci TCP zásuvky, naslouchá na definovaném portu a navazuje spojení. Pro každé spojení vytváří instanci třídy THttpSession.
THttpSession	Provádí příjem, odeslání odpovědi a ukončení spojení. Obsahuje dva vnořené objekty THttpRequest a THttpResponse. Po ukončení spojení instance THttpSession zaniká.
THttpRequest	Zajišťuje zpracování (překlad) HTTP dotazu z textové podoby do binární. Pomocí této třídy lze získat informace o požadavku.
THttpResponse	Provádí překlad HTTP odpovědi z binární podoby do textové.

Pomocí této třídy se vytváří odpověď.



Aplikace využívající knihovny Http se musí postarat o:

- Inicializaci TCP/IP zásobníku (viz. funkce **NetOpenStack**, **NetCloseStack** jednotky CoInet)
- Inicializaci instance třídy **THttpServer** (viz. kapitola 5.4.2.1)
- Obsluhu jednotlivých požadavků (viz. funkce **DataProvider** v kapitole 5.4.1.1)
- Periodické volání metody **Tick** třídy **THttpServer** v rámci některého z procesů aplikace.

Knihovna definuje několik pomocných funkcí a tříd:

- Funkce **HttpGetPathSegment** a **HttpDispatchSegment** slouží ke zpracovaní cesty k dokumentu.
- Třída **THttpQueryParser** usnadňuje zpracování parametrů dotazu, tj. textu za '?' v URI.

3.5. Jak ladit aplikaci s kartou IOETH01

Ladění aplikací využívajících karty IOETH01 přímo na PC je poněkud komplikovanější, než ladění běžné aplikace, protože v současné době není k dispozici softwarový emulátor této karty.

Při ladění je nutné mít v ISA slotu zasunutou kartu PC-Kit (případně LPT-Kit na paralelním portu) a k ní připojenou desku IOETH01. Ethernetová karta může být

připojena do místní lokální sítě, nebo pomocí překříženého kabelu k jinému počítači. Klient (WWW prohlížeč) by měl běžet na jiném počítači, než na kterém je spuštěna laděná aplikaci, protože Microsoft Windows 95, 98 a ME nepřidělují DOSovým aplikacím běžícím na pozadí žádný procesorový čas.

V prohlížeči stránek (např. v Microsoft Internet Explorer) zadejte adresu ve formátu:

http://*adresa_kita[:port]*/*[cesta_k_dokumentu]*

(např. http://192.168.1.10:1500/index.html)

Pokud jste nezměnili v nastavení třídy THttpServer číslo portu (viz. kapitola 5.4.1.5), který má implicitní hodnotu 80, není potřeba číslo portu uvádět.

3.6. Zvolení IP adresy jednotky KIT

Pokud chcete připojit kartu IOETH01 do místní lokální sítě musíte se seznámit s politikou přidělování adres v místní síti. Každá stanice musí mít přidělenou jednoznačnou IP adresu. Pokud si nejste jisti, že vámi zvolená IP adresa je unikátní, požádejte správce sítě o přidělení volné adresy nebo rozsahu adres.

IP adresu počítače a masku sítě zjistíte např. pomocí programu winipcfg.exe (Microsoft Windows 95, 98 a ME) nebo ipconfig.exe (Microsoft Windows 2000 a XP).

Příklad:

V sítí máte připojené tři počítače. Pomocí programu ipconfig.exe jste zjistil, že IP adresy jednotlivých počítačů jsou 192.168.1.1, 192.168.1.2 a 192.168.1.3 a maska sítě je 255.255.255.0. Jednotce KIT můžete tedy přidělit adresu z rozsahu 192.168.1.4 – 192.168.1.254 a masku 255.255.255.0.

4. Konstanty a jednoduché typy

4.1. Konstanty

4.1.1. Chybové kódy HTE_xxx

Součástí každé odpovědi na HTTP dotaz je stavový řádek obsahující chybový kód a upřesňující text, např.: „HTTP1/0 201 Created“. Specifikace protokolu http v RFC-1945 a RFC-2068 definují množinu návratových kódu, které jsou uvedeny v tabulce níže. Chybové kódy jsou rozděleny do čtyřech skupin:

Rozsah	Popis třídy
200-299	Tato třída stavových kódů indikuje úspěšné přijetí HTTP požadavku
300-399	Tato třída stavových kódů indikuje, že pro splnění požadavku musí klient provést další operace. Např. chyba 301 Moved Permanently - oznamuje trvalé přesunutí dokumentu na jiné místo.
400-499	Tuto třídu chyb server využívá k oznámení nějaké chyby na straně klienta. Např. 400 Bad Request – oznamuje neplatný formát/syntaxi požadavku.
500-599	Tato třída chyb oznamuje chybu serveru nebo jeho neschopnost provést zadaný požadavek.

Většina jednoduchých aplikací si pravděpodobně vystačí s návratovými kódy 200 OK a 404 Not Found. Více informací lze nalézt v RFC-1945 a RFC-2068 (chybové kódy označené *).

Kód	Identifikátor
200	HTE_OK
201	HTE_CREATED
202	HTE_ACCEPTED
203	HTE_NON_AUTH_INFORMATION *
204	HTE_NO_CONTENT
205	HTE_RESET_CONTENT *
206	HTE_PARTIAL_CONTENT *
300	HTE_MULTIPLE_CHOICES
301	HTE_MOVED_PERMANENTLY
302	HTE_MOVED_TEMPORARILY
303	HTE_SEE_OTHER *
304	HTE_NOT_MODIFIED
305	HTE_USE_PROXY *
400	HTE_BAD_REQUEST
401	HTE_UNAUTHORIZED
402	HTE_PAYMENT_REQURIED *
403	HTE_FORBIDDEN

404 HTE_NOT_FOUND
405 HTE_METHOD_NOT_ALLOWED *
406 HTE_NOT_ACCEPTABLE *
407 HTE_PROXY_AUTH_REQUIRED *
408 HTE_REQUEST_TIMEOUT *
409 HTE_CONFLICT *
410 HTE_GONE *
411 HTE_LENGTH_REQUIRED *
412 HTE_PRECONDITION_FAILED *
413 HTE_REQ_ENTITY_TOO_LARGE *
414 HTE_REQ_URI_TOO_LONG *
415 HTE_UNSUPP_MEDIA_TYPE *

500 HTE_INTERNAL_SERVER_ERROR
501 HTE_NOT_IMPLEMENTED
502 HTE_BAD_GATEWAY
503 HTE_SERVICE_UNAVAILABLE
504 HTE_GATEWAY_TIMEOUT *
505 HTE_HTTP_VERSION_NOTSUPP *

4.2. Jednoduché typy

4.2.1. Typ THttpMethod

```
THttpMethod = (
  mtdGet,
  mtdHead,
  mtdPost,
  mtdPut,
  mtdDelete
);
```

Výčtový typ **THttpMethod** identifikuje typ HTTP dotazu klienta. Viz. položka **Method** třídy **THttpRequest** (kapitola 5.6.1.1)

4.2.2. Typ THttpDataProvider

```
THttpDataProvider = function(  ASession: PHttpSession ): Boolean;
```

THttpDataProvider je procedurální typ definující funkci pro obsluhu HTTP požadavku. Viz. položky **DataProvider** tříd **THttpServer** a **THttpSession** (kap. 5.4.1.1)

4.2.3. Typ THttpErrorHandler

```
THttpErrorHandler = procedure( ASession: PHttpSession;
                      const AText: string );
```

THttpErrorHandler je procedurální typ definující funkce pro obsluhu chyb HTTP protokolu. Viz. položka **ErrorHandler** třídy **THttpServer** (kap. 5.4.1.2)

4.2.4. Typ THttpQueryParam

```
PHttpQueryParam = ^THttpQueryParam;
THttpQueryParam = record
  Name  : PChar;
  Value : PChar;
end;

TAHttpQueryParam = array[0..255] of THttpQueryParam;
```

Typ **THttpQueryParam** je reprezentuje strukturu položky tabulky parametrů specifikovaných v HTTP dotazu. Viz třída **THttpQueryParser** v kapitole 4.2.4.

4.2.5. Typ THttpPathSegment

```
PHttpPathSegment = ^THttpPathSegment;
THttpPathSegment = record
  Segment : PChar;
  Id      : Integer;
end;
```

Položka tabulky segmentů (viz. funkce **HttpGetPathSegment** v kapitole 5.2).

4.2.6. Typ THttpDispatchItem

```
PHttpDispatchItem = ^THttpDispatchItem;
THttpDispatchItem = record
  Segment    : PChar;
  Dispatcher : THttpSegmentDispatcher;
end;
```

Položka tabulky segmentů (viz. funkce **HttpDispatchSegment** v kapitole 5.3).

4.2.7. Typ THttpSegmentDispatcher

```
THttpSegmentDispatcher = function( ASession: PHttpSession;
  APath: PChar; AIndex: Integer ): Boolean;
```

Prototyp funkce obsluhy požadavku podle tabulky segmentů (viz. funkce **HttpDispatchSegment** v kapitole 5.3)

5. Funkce a procedury

5.1. Funkce HttpGetTime

Funkce **HttpGetTime** vrací aktuální čas ve spakovaném tvaru.

```
function HttpGetTime: Longint;
```

Parametry:

Funkce nemá žádné parametry.

Návratové hodnoty:

Funkce **HttpGetTime** vrací aktuální čas ve spakovaném tvaru.

Poznámky:

Funkce **HttpGetTime** používá funkce jednotky DOS - GetTime, GetDate a PackTime.

5.2. Funkce HttpGetPathSegment

Funkce **HttpGetPathSegment** provádí zpracování jednoho segmentu cesty k dokumentu podle zadané překladové tabulky. Funkce hledá segment cesty v tabulce a vrací jeho identifikátor.

```
function HttpGetPathSegment( var APath: PChar;
          ATable: PHttpPathSegment; var AIndex: Integer ): Integer;
```

Parametry:

APath	Ukazatel na cestu k dokumentu.
ATable	Ukazatel na první položku pole struktur THttpPathSegment, tj. tabulky definující všechny možné dokumenty a složky na jedné úrovni cesty.
AIndex	Ukazatel na proměnnou, která bude nastavena na index v názvu segmentu, viz. dále.

Návratové hodnoty:

V případě, že segment cesty se shoduje s alespoň jednou položkou v tabulce, pak funkce vrací identifikátor segmentu uvedený v tabulce ATable, v opačném případě vrací hodnotu –1.

Poznámky:

Funkced **HttpGetPathSegment** usnadňuje zpracování cesty k dokumentu zadané na příkazové řádce HTTP dotazu. Tuto cestu lze získat z položky **Path** třídy **THttpRequest**.

Jedním z parametrů funkce je ATable, což je ukazatel na začátek tabulky (pole) položek **THttpPathSegment**. Tabulka obsahuje seznam položek na jedné úrovni cesty (lze si představit jako jednu úroveň adresáře). Tabulku můžeme nadefinovat např. jako typovou konstantu:

```
const
  ARootFolder: array[0..4] of THttpPathSegment =
  (
    ( Segment: 'adresar1/';   Id: 10 ),
      Segment: 'index.html';  Id: 20 ),
      Segment: 'abc?';        Id: 30 );
      Segment: '';            Id: 40 ),
      Segment: nil;           Id: -1 )
  );
```

Tabulka musí vždy končit položkou s hodnotami Segment = **nil** a Id = -1.

Pokud je na konci názvu segmentu uveden znak "/", pak se jedná o složku. V opačném případě jde o dokument. Otazník v názvu segmentu označuje místo, kde se očekává celočíselná kladná konstanta v rozsahu 0 až 32767. Tato konstanta je předaná pomocí parametru funkce AIndex. **Při porovnávání se rozlišují malá a velká písmena**.

Když máme takto připravenou tabulku, můžeme v obsluze požadavku volat funkci **HttpGetPathSegment** např. takto:

```
var
  Path  : PChar;    { Cesta k dokumentu                          }
  Index : Integer;  { Index v názvu dokumentu                    }
  Id    : Integer;  { Identifikátor dokumentu 10, 20, 30, 40, -1..}

  { Ukazatel na text cesty získáme ze struktury požadavku }
  Path := Request.Path;

  { Zpracování první úrovně cesty }
  Id := HttpGetPathSegment( Path, @ARootFolder, Index );
```

Proměnná Id bude po zavolání funkce **HttpGetPathSegment** obsahovat identifikátor segmentu uvedený v tabulce, příp. –1 pokud se cesta neshoduje s žádnou položkou v tabulce. Proměnná APath bude ukazovat za zpracovanou část cesty, pro případné opakované volaní funkce **HttpGetPathSegment** pro další úroveň.

V následující tabulce jsou uvedeny výsledky funkce **HttpGetPathSegment** ve výše uvedeném příkladu pro různé hodnoty vstupní proměnné APath.

Path	Id	Index
/	40	nedefinováno(-1)
/adresar1/dokumentX	10	nedefinováno(-1)
/abc123	30	123
/INDEX.html	-1	nedefinováno (-1)

5.3. Funkce HttpDispatchSegment

Funkce **HttpDispatchSegment** provádí zpracování jednoho segmentu cesty k dokumentu podle zadané překladové tabulky. Funkce hledá segment cesty v tabulce a volá přidruženou funkci, která provede zpracování požadavku.

```
function HttpDispatchSegment( APath: PChar;
      ATable: PHttpDispatchItem; ASession: PHttpSession ): Boolean;
```

Parametry:

APath	Ukazatel na cestu k dokumentu.
ATable	Ukazatel na první položku pole struktur THttpDispatchItem, tj. tabulky definující všechny možné dokumenty a složky na jedné úrovni cesty.
ASession	Ukazatel na relace, tj. instanci třídy THttpSession, která je předána obslužné rutině Dispatcher v případě nalezení shodné cesty segmentu v tabulce.

Návratové hodnoty:

V případě, že segment cesty se shoduje s alespoň jednou položkou v tabulce, pak funkce vrací návratovou hodnotu zavolané funkce Dispatcher z téže položky tabulky. Jinak vrací hodnotu True.

Poznámky:

Funkced **HttpDispatchSegment** usnadňuje zpracování cesty k dokumentu zadané na příkazové řádce HTTP dotazu. Tuto cestu lze získat z položky **Path** třídy **THttpRequest**.

Jedním z parametrů funkce je ATable, což je ukazatel na začátek tabulky (pole) položek **THttpDispatchSegment**. Tabulka obsahuje seznam položek na jedné úrovni cesty (lze si představit jako jednu úroveň adresáře). Tabulku můžeme nadefinovat např. jako typovou konstantu:

```
const
  ARootFolder: array[0..4] of THttpPathSegment =
  (
    ( Segment: 'adresar1/';   Dispatcher: DispAdresar1 ),
      Segment: 'index.html';  Dispatcher: DispIndexHtml ),
      Segment: 'abc?';        Dispatcher: DispAbc );
      Segment: '';            Dispatcher: DispRoot ),
      Segment: nil;           Dispatcher: DispError )
  );
```

Tabulka musí vždy končit položkou s hodnotami Segment = **nil**.

Pokud je na konci názvu segmentu uveden znak /, pak se jedná o složku. V opačném případě jde o dokument. Otazník v názvu segmentu označuje místo, kde se očekává

celočíselná kladná konstanta v rozsahu 0 až 32767. Tato konstanta je předaná pomocí parametru AIndex. **Při porovnávání položek tabulky a textu cesty je rozlišují malá a velká písmena**.

Funkce Dispatcher musí mít následující prototyp:

```
THttpSegmentDispatcher = function( ASession: PHttpSession;
  APath: PChar; AIndex: Integer ): Boolean;
```

Parametr **ASession** je shodný se stejnojmenným parametrem funkce **HttpDispatchSegment**. Parametr **APath** ukazuje na další nezpracovaný segment na cestě k dokumentu (funkci **HttpDispatchSegment** je možné volat rekurzivně) a parametr **AIndex** obsahuje konstantu v rozsahu 0 až 32767 (viz. odstavec výše).

5.4. Třída THttpServer

5.4.1. Položky třídy THttpServer

5.4.1.1. DataProvider

```
DataProvider : THttpDataProvider;
```

Položka **DataProvider** určuje funkci, která je zavolána vždy, když jsou na HTTP serveru požadována data. Úkolem této funkce je zpracovat dotaz a vyplnit data odpovědi. Položku **DataProvider** je potřeba nastavit ihned po vytvoření instance třídy **THttpServer**, implicitně je nastavená na hodnotu **nil**.

THttpDataProvider je procedurální typ deklarovaný následovně:

```
THttpDataProvider = function ( ASession: PHttpSession ): Boolean;
```

Parametr ASession je ukazatel na instanci relace. Požadavek je přístupný přes položku ASession^**.Request** a odpověď skrze položku ASession^**.Response**.

Funkce **DataProvider** je periodicky volána tak dlouho dokud nevrátí hodnotu True. Pak teprve automat serveru považuje relaci za ukončenou a odešle připravená data. Tento mechanismus umožňuje vytvoření dalšího vnořeného automatu pro zpracování dotazu. Viz kapitola 6. V běžných případech, kdy je možné připravit data odpovědi ihned, vrací funkce hodnotu True vždy.

Pokud funkce **DataProvider** nevyplní data odpovědi, pak je jako odpověď odeslána pouze řádka se položkami THttpResponse.**StatusCode** a THttpResponse.**ReasonString**. Jestliže funkce dotaz nezpracuje a jednoduše vrátí hodnotu True. Poté server sám odešle implicitní odpověď s kódem 404 Not Found, oznamující, že požadovaný dokument nebyl nalezen.

Prvními kroky uvnitř této funkce je obvykle rozvětvení podle typu požadavku (tj. položky Request.Method). Dalším krokem je zpracování cesty k požadovanému dokumentu (tj. zpracování Request.Path) pomocí funkcí **HttpGetPathSegment** nebo **HttpDispatchSegment**. V posledním kroku se vyplní návratový kód odpovědi (viz. THttpResponse.**StatusCode**) a úplně nakonec data odpovědi (viz. THttpResponse.**BeginEntity** a THttpResponse.**EndEntity**).

5.4.1.2. ErrorHandler

```
ErrorHandler : THttpErrorHandler;
```

Položka **ErrorHandler** určuje funkci, která je volaná vždy, když v serveru vznikne abnormální událost, např. chyba TCP zásuvky, interní chyba serveru, vypršení časového limitu na relaci apod. Pokud je položka nastavena na hodnotu **nil**, pak se obsluha chyb nevolá. Položku **ErrorHandler** je potřeba nastavit ihned po zavolání vytvoření instance třídy **THttpServer**, implicitně je nastavená na hodnotu **nil**.

THttpErrorHandler je procedurální typ deklarovaný následovně:

```
THttpErrorHandler = procedure ( ASession: PHttpSession;
                      const AText: string );
```

Parametr ASession ukazatel na instanci relace, ve které chyba vznikla. Pokud chyba není svázána s konkrétní relací, pak je tento parametr nastaven na hodnotu **nil**.
Parametr AText obsahuje textový popis chyby. Tento je možné vypsat na displej (v případě ladící verze aplikace) nebo jej uložit do archívu, který je možné nějakým způsobem později zobrazit. Textové popisy nejsou delší než 50 znaků.

5.4.1.3. AcceptFilter

```
AcceptFilter : THttpAcceptFilter;
```

Položka **AcceptFilter** definuje funkci, která filtruje příchozí spojení podle adresy. Funkce umožňuje selektivně zamezit některým stanicím v sítí přístup k HTTP serveru. Položku **AcceptFilter** je potřeba nastavit ihned po vytvoření instance třídy **THttpServer**, implicitně je nastavená na hodnotu **nil**, což znamená, že jsou povolena spojení s libovolnou stanicí.

THttpAcceptFilter je procedurální typ deklarovaný následovně:

```
THttpAcceptFilter = function( AAdress: PTcpAddress ): Boolean;
```

Typ **TTcpAddress** definuje knihovna CoTCP a má následující strukturu:

```
TTcpAddress =
record
  Size    : Byte;
  Address : TIpAddress;  { Longint }
  Port    : Word;
end;
```

Na základě předané adresy, může funkce rozhodnout, zda povolí navázání spojení, nebo jej zamítne. Pokud má být spojení navázáno, musí funkce vrátit hodnotu True a v opačném případě False.

5.4.1.4. ConnTimeout

```
ConnTimeout: TCoTime;  { Longint }
```

Položka **ConnTimeout** specifikuje čas rezervovaný na zpracování požadavku od navázání spojení. Tento čas se zadává v milisekundách a po inicializaci instance serveru je nastaven na implicitní hodnotu 10000 (10s). Po vypršení zadaného času je relace neprodleně ukončena a spojení uzavřeno. Pokud je zadána hodnota 0, časové omezení na provedení relace se neuplatní. Položku **ConnTimeout** lze nastavit ihned po vytvoření instance třídy **THttpServer**. V době kdy server běží, nelze tuto hodnotu měnit.

5.4.1.5. Port

```
Port : Word;
```

Položka port udává číslo TCP portu HTTP serveru. Implicitně je tato položka nastavena na standardní hodnotu 80. Položku **Port** lze nastavit ihned po vytvoření instance třídy **THttpServer**. V době kdy server běží, se změna této hodnoty nijak neuplatní.

5.4.2. Metody třídy THttpServer

5.4.2.1. Konstruktor Init

Konstruktor **Init** inicializuje instanci třídy **THttpServer**.

```
constructor THttpServer.Init( AMaxSessions: Integer;
              const ANetStackId: string;
              AMaxReqSize, AMaxRespSize: Word );
```

Parametry:

AMaxSessions	Maximální počet relací probíhajících současně.
ANetStackId	Identifikátor TCP/IP zásobníku, na který se má HTTP server navázat. Tento parametr se specifikuje ve funkci **NetOpenStack** knihovny CoInet při vytváření TCP/IP zásobníku.
AMaxReqSize	Maximální velikost přijatého dotazu v bajtech.
AMaxRespSize	Maximální velikost odesílané odpovědi v bajtech.

Poznámky:
Kontruktor **Init** iniciazuje třídu **THttpServer** a nastaví položky instance na implicitní hodnoty. Po vytvoření instance je nutné ručně nastavit položky **DataProvider** a **ErrorHandler**, a případně číslo portu serveru (implicitně nastavené na standardní hodnotu 80).

Parametry AMaxReqSize a AMaxRespSize by měly být nastaveny podle potřeb konkrétní aplikace. Jako minimální hodnoty doporučujeme 1024, tedy 1kB jak na požadavek, tak na odpověd.

5.4.2.2. Destruktor Done

Konstruktor **Done** uvolní instanci třídy **THttpServer** z paměti.

```
destructor THttpServer.Done;
```

Parametry:

Destruktor nemá žádné parametry.

Poznámky:

Destruktor Done třídy **THttpServer** by neměl být volán do té doby než metoda **Tick** vrátí hodnotu True, v opačném případě nemusí dojít ke správnému ukončení probíhajících relací.

5.4.2.3. Metoda Stop

Metoda **Stop** ukončí činnost HTTP serveru.

```
procedure THttpServer.Stop;
```

Parametry:

Metoda **Stop** nemá žádné parametry.

Poznámky:

Metoda **Stop** nastaví interní příznak, který je pravidelně testován v rámci metody **Tick**. Pokud je tento příznak nastaven, server se snaží bez prodleně ukončit svou činnost. V okamžiku kdy je server ukončen, metoda **Tick** vrátí hodnotu True. Vícenásobné volání metody **Stop** nemá žádný efekt.

5.4.2.4. Metoda Start

Metoda **Start** provede spuštění HTTP serveru.

```
procedure THttpServer.Start;
```

Parametry:

Metoda **Start** nemá žádné parametry.

Poznámky:

Metoda **Start** nastaví interní příznak, který je pravidelně testován v rámci metody **Tick**. Pokud byla činnost serveru pozastavena předchozím voláním metody **Stop**, volání metody **Start** opět server spustí. V čase mezi zastavením serveru a jeho znovuspuštěním vrací metoda **Tick** hodnotu True.

5.4.2.5. Metoda Tick

Metoda **Tick** provede jeden krok automatu serveru.

```
function THttpServer.Tick: Boolean;
```

Parametry:

Metoda **Tick** nemá žádné parametry.

Návratové hodnoty:

Metoda **Tick** vrací True, pokud je server zastaven, v opačném případě vrací False.

Poznámky:

Metoda **Tick** „pohaní“ celý HTTP server. Metodu **Tick** je potřeba periodicky volat např. v hlavní smyčce programu nebo v rámci hlavní smyčky některého z procesů aplikace. Pokud je potřeba z nějakého důvodu předčasně ukončit činnost serveru, je nutné nejprve zavolat metodu Stop a pak vyčkat než metoda **Tick** vrátí hodnotu True. Poté je možné bezpečně uvolnit instanci serveru. Pokud uvolníme instanci serveru dříve, nemusí být právě probíhající transakce dokončeny správně.

5.5. Třída THttpSession

Instance třídy **THttpSession** jsou vytvářeny a rušeny automaticky se vznikajícími a zanikajícími spojeními s klienty. Ukazatel na instanci je k dispozici při obsluze požadavku a obsluze chyby (viz položka THttpServer.**DataProvider** a THttpServer.**ErrorHandler**)

5.5.1. Položky třídy THttpSession

5.5.1.1. Request

```
Request : THttpRequest;
```

Položka **Request** je vnořená instance třídy **THttpRequest**. Pomocí této položky lze přistupovat k parametrům HTTP dotazu (viz. třída **THttpRequest** v kapitole 5.6)

5.5.1.2. Response

```
Response : THttpResponse;
```

Položka **Response** je vnořená instance třídy **THttpResponse**. Pomocí této položky lze přistupovat k parametrům HTTP odpovědi (viz. třída **THttpResponse** v kapitole 5.7)

5.5.1.3. ClientAddr

```
ClientAddr : TTcpAddress;
```

Položka **ClientAddr** obsahuje adresu a port klienta specifický pro tuto relaci. Typ **TTcpAdress** je definován v knihovně CoTCP a má následující strukturu:

```
TTcpAddress =
record
  Size    : Byte;
  Address : TIpAddress;  { Longint }
  Port    : Word;
end;
```

Položka ClientAddr může být užitečná např. v případě, kdy chceme na základě adresy klienta rozlišit poskytované informace. Např. některé vybrané stanice (autorizované stanice) mohou obdržet více informací, případně nastavovat parametry, které jiné stanice nemohou.

5.5.1.4. DataProvider

```
DataProvider : THttpDataProvider;
```

Položka **DataProvider** určuje funkci, která je volána, když jsou požadována data odpovědi. Úkolem této funkce je zpracovat dotaz a vyplnit data odpovědi. Položka **DataProvider** je automaticky nastavena při vytvoření relace na hodnotu stejnojmenné položky instance serveru **THttpServer**. Pokud jsme schopni na požadavek klienta odpovědět okamžitě, není potřeba položku **DataProvider** měnit. Jestliže nejsme schopni odpovědět ihned pak funkce **DataProvider** vrátí False a předtím můžeme další zpracování požadavku přesměrovat na jinou funkci. Viz kapitola 6. Úplný popis této je uveden v kapitola 5.4.1.1.

5.5.1.5. ErrorHandler

```
ErrorHandler : THttpErrorHandler;
```

Položka **ErrorHandler** určuje funkci, která je volaná vždy, když v serveru vznikne abnormální událost, např. chyba TCP zásuvky, interní chyba serveru, vypršení časového limitu na relaci apod. Pokud je položka nastavena na hodnotu **nil**, pak se obsluha chyb nevolá. Položku **ErrorHandler** je nastavena při vytvoření relace na stejnojmennou hodnotu instance serveru **THttpServer**. Tuto položku není potřeba měnit.Úplný popis této položky je uveden v kapitole 5.4.1.2.

5.5.1.6. Context

```
Context : Pointer;
```

Položku **Context** může využít funkce **DataProvider** pro uložení libovolných dat potřebných pro zpracování odpovědí. Položka je vždy při vytvoření relace nastavena na implicitní hodnotu **nil**. Kromě tohoto implicitního nastavení knihovna HTTP tuto položku nenastavuje ani jinak nezpracovává. Položku **Context** lze využít v případě, když nejsme schopni na požadavek klienta odpovědět ihned a v rámci funkce **DataProvider** implementujeme automat. Viz kapitola 6.

5.6. Třída THttpRequest

5.6.1. Položky třídy THttpRequest

5.6.1.1. Method

```
Method : THttpMethod;

THttpMethod = (
  mtdGet,
```

```
  mtdHead,
  mtdPost,
  mtdPut,
  mtdDelete
);
```

Položka **Method** určuje typ metody použité na příkazové řádce dotazu klienta. Nejběžnější metody jsou GET a POST. Metoda GET je určena pro získání dat ze serveru a metoda POST pro odeslání dat na server. Více v RFC-1945.

5.6.1.2. Path

```
Path : PChar;
```

Položka **Path** obsahuje cestu k požadovanému dokumentu na serveru. Tuto položku je možné zpracovat pomocí funkcí **HttpGetPathSegment** nebo **HttpDispatchSegment**. Položka má následující strukturu:

“hostitel/cesta” nebo zjednodušený tvar “/cesta“ např.

“www.sofcon.cz/index.html”
“/index.html”
“192.168.1.1/tabulky/teploty“
“/”

Ve všech případech je obsahuje cesta alespoň jedno lomítko. Před prvním lomítkem může být IP adresa nebo název domény. Za prvním lomítkem je absolutní cesta k dokumentu. Přesnější specifikace této položky je uvedena v RFC-1945.

5.6.1.3. Query

```
Query : PChar;
```

Položka **Query** obsahuje parametry dotazu uvedené za znakem ‘?‘ příkazové řádky sestavené klientem. Struktura těchto parametrů je obecně nedefinovaná. Za znakem ‘?’ se může nacházet libovolný text. Obvykle je však struktura následující:

“parametr1=hodnota1&parametr2=hodnota2&…”.

Tento formát používají např. HTML formuláře. Pro zpracování takového dotazu lze použít třídu **THttpQueryParser** (viz kapitola 5.9).

Pokud dotaz neobsahuje parametry, pak položka **Query** obsahuje hodnotu **nil**.

5.6.1.4. Version

```
Version : PChar;
```

Položka **Version** obsahuje verzi HTTP protokolu. Současná verze knihovny podporuje pouze protokol verze 1.0 a některé prvky protokolu verze 1.1. Hodnota položky **Version** je vždy text “HTTP/1.0“.

5.6.1.5. Date

```
Date: Longint;
```

Položka **Date** je součástí všeobecného záhlaví požadavku. Obsahuje datum a čas okamžiku, kdy byl požadavek sestaven a odeslán. Položka **Date** je datum a čas ve spakovaném tvaru. Pro rozbalení použijte funkci **UnpackTime** jednotky Dos. Pokud záhlaví požadavku neobsahuje tuto informaci, pak položka **Date** obsahuje hodnotu 0. Tato položka je pouze informativní a lze ji ignorovat.

5.6.1.6. Host

```
Host: PChar;
```

Položka **Host** obsahuje adresu a port serveru. Více v RFC-2068. Pokud tato informace v záhlaví požadavku nebyla uvedena, obsahuje hodnotu **nil**.

5.6.1.7. IfModified

```
IfModified : Longint;
```

Položka **IfModified** je používána společně s metodou GET k jejímu podmínění. Pokud požadovaný obsah nebyl modifikován později než udává tato položka, měl by server odpovědět chybovým kódem 304 (HTE_NOT_MODIFIED). V opačném případě musí odpovědět, jako kdyby položka v záhlaví požadavku uvedena nebyla. Položka **IfModified** obsahuje datum a čas ve spakovaném tvaru. Pokud tato informace v záhlaví požadavku uvedena nebyla, položka obsahuje hodnotu 0. Více v RFC-1945 (If-Modified-Since)

5.6.1.8. Referer

```
Referer : PChar;
```

Položka **Referer** obsahuje cestu ke stránce, na které byl získán odkaz na právě požadovanou stránku. Tato položka je pouze informativní a lze ji ignorovat. Více v RFC-1945.

5.6.1.9. UserAgent

```
UserAgent: PChar;
```

Položka **UserAgent** identifikuje klienta, který vytvořil požadavek. Např. Microsoft Explorer 6.0 generuje této popis:

```
„Mozilla/4.0 (compatible; MSIE6.0; Windows 98; MSNATLAS01.CZ)".
```

Pokud tato informace není součástí záhlaví požadavku, obsahuje položka **UserAgent** hodnotu **nil**. Tato položka je pouze informativní a lze ji ignorovat.Více v RFC-1945.

5.6.1.10. ContentLength

```
ContentLength : Word;
```

Položka **ContentLength** obsahuje délku dat předané entity. Pokud data entity v

požadavku uvedena nebyla, obsahuje tato položka hodnotu 0.

5.6.1.11. ContentType

```
ContentType : PChar;
```

Položka **ContentType** je součástí záhlavní entity. Obsahuje typ dat v odeslané entity. Tato položka má následující strukturu: “typ**:**podtyp“ např. “text/html”, “image/jpeg” , ”image/gif” apod. Pokud tato položka v záhlaví entity nebyla uvedena, pak obsahuje hodnotu **nil**. Více v RFC-1945, RFC-1700, RFC-1521

5.6.1.12. Accept

```
Accept : PChar;
```

Položka **Accept** omezuje množinu typu dat, které může server odeslat v rámci odpovědi. Hodnota položky **Accept** má následující strukturu: “typ:podtyp, typ:podtyp, typ:podtyp, …”. Pokud tato informace v záhlaví požadavku nebyla uvedena, položka obsahuje hodnotu **nil**. Pokud server odesílá pouze odpovědi typu text/html, text/xml apod., lze tuto položku ignorovat. Více v RFC-2068.

5.6.1.13. Content

```
Content : PChar;
```

Položka **Content** ukazuje na data entity. Pokud data entity nejsou součástí požadavku, pak tato položka obsahuje hodnotu **nil**. Data entity jsou zakončena vždy znakem #0. Pokud však data entity mohou obsahovat tento ukončovací znak, lze délku zjistit pomocí položky **ContentLength**.

5.7. Třída THttpResponse

5.7.1. Položky třídy THttpResponse

5.7.1.1. Pragma

```
Pragma : PChar;
```

Položka **Pragma** je implicitně nastavena na hodnotu “no-cache”. Toto nastavení instruuje klienta příp.proxy servery, aby neukládaly data odpovědi do svých vyrovnávacích pamětí. Pokud chcete toto nastavení změnit přiřaďte této položce hodnotu **nil**.

5.7.1.2. StatusCode

```
StatusCode : Word;
```

Položka **StatusCode** obsahuje chybový kód odpovědi (viz. kapitola 4.1.1). Položka je implicitně nastavena na hodnotu 404 (Not Found) – dokument nenalezen.

5.7.1.3. ReasonString

```
ReasonString : PChar;
```

Položka **ReasonString** upřesňuje chybový kód odpovědi, může obsahovat textový popis chyby, který obvykle klient (Internet Explorer) zobrazí při selhání načítání stránky. Implicitně je položka **ReasonString** nastavena na hodnotu nil, a jestliže položka **StatusCode** obsahuje jeden, ze standardních kódu z kapitoly 4.1.1, je **ReasonString** před odesláním odpovědi automaticky vyplněna standardním textem.

5.7.1.4. Location

```
Location : PChar;
```

Položka **Location** obsahuje přesnou absolutní cestu k vyžádanému dokumentu serveru. Tuto položku není potřeba nastavovat, pokud server nevrací chybový kód ze skupiny 3XX (přesměrování). Implicitně je položka **Location** nastavena na hodnotu **nil** a tudíž není do záhlaví odpovědi zahrnuta.Viz RFC-1945.

5.7.1.5. Server

```
Server : PChar;
```

Položka Server identifikuje software serveru. Implicitně je tato položka nastavena na hodnotu **nil**. Můžete zde indikovat např. název, verzi vašeho serveru. Viz RFC-1945. Pokud je položka **Server** nastavena na hodnotu **nil**, pak není do záhlaví odpovědi zahrnuta.

5.7.1.6. Allow

```
Allow : PChar;
```

Položka **Allow** obsahuje seznam metod aplikovatelných na požadovaný dokument. Jedná se o seznam metod oddělených čárkou, např.: “GET, HEAD“. Implicitně je tato položka nastavena na hodnotu **nil** a obvykle ji není potřeba měnit. Přesnější informace v RFC-1945. Pokud je položka **Allow** nastavena na hodnotu **nil**, pak není do záhlaví odpovědi zahrnuta.

5.7.1.7. ContentEncoding

```
ContentEncoding : PChar;
```

Položka **ContentEncoding** slouží k určení typu kódování posílaného obsahu. Implicitně je tato položka nastavena na **nil**, což znamená, že žádné kódování použito není. Tato položka je primárně používána k určení typu komprese. Pokud nepoužíváte kompresi dat, pak tuto položku není nutné měnit. Viz RFC-1945.

5.7.1.8. ContentType

```
ContentType : PChar;
```

Položka **ContentType** určuje typ přenášených dat v obsahu odpovědi. Tato položka je

implicitně nastavena na hodnotu "text/html" a pokud odpověď tvoří HTML stránka, není potřeba tuto položku měnit. Struktura položky je následující: "typ/podtyp". Viz. RFC-1945. Pokud je položka **ContentType** nastavena na hodnotu **nil**, není v rámci záhlaví odpovědi odesílána.

Jestliže odpověď tvoří data ve formátu XML, nastavte tuto položku na hodnotu "text/xml". V případě, že odesíláte prostý text použijte hodnotu "text/plain". Více informací o typech a podtypech je uvedeno v RFC-1521 a RFC-1700.

5.7.1.9. Expires

```
Expires : Longint;
```

Položka **Expires** udává datum a čas, kdy vyprší platnost předaných dat. Klient může tuto informaci využít např. k řízení vyrovnávací paměti (cache). Implicitně je tato položka nastavena na hodnotu 0, což znamená, že není v rámci záhlaví odpovědi odesílána. Hodnota položky je datum a čas ve spakovaném tvaru. Tuto položku není nutné v běžných případech nastavovat. Viz RFC-1945.

5.7.1.10. LastModified

```
LastModified : Longint;
```

Položka **LastModified** udává datum a čas poslední modifikace předávaného dokumentu. Klient může tuto informaci využít např. k řízení vyrovnávací paměti (cache). Implicitně je tato položka nastavena na hodnotu 0, což znamená, že není v rámci záhlaví odpovědi odesílána. Hodnota položky je datum a čas ve spakovaném tvaru. Tuto položku není nutné v běžných případech nastavovat. Viz RFC-1945

5.7.1.11. Content

```
Content : THttpResponseText;
```

Položka **Content** je vnořená instance třídy **THttpResponseText**, což je buffer pro data odpovědi. Do tohoto bufferu je zapisováno jak záhlaví odpovědi (tzn. obsah položek uvedených v kapitolách 5.7.1.1 až 5.7.1.10), tak i uživatelská data (entita).

Třídy **THttpResponseText** je popsána v kapitole 5.8.

5.7.2. Metody třídy THttpResponse

Třída **THttpResponse** nabízí celkem čtyři metody, které se volají v rámci kódu obsluhy požadavku. Metody slouží k přípravě záhlaví odpovědi.

5.7.2.1. Metoda AddStatusLine

Metoda **AddStatusLine** uloží do bufferu odpovědi stavový řádek s chybovým kódem.

```
procedure THttpResponse.AddStatusLine;
```

Parametry:

Metoda nemá žádné parametry.

Poznámky:

Metoda **AddStatusLine** vymaže buffer odpovědi a zapíše do něj stavový řádek obsahující složený z položek **StatusCode** a **ReasonString**. Např.

HTTP1/0 200 OK

Metoda **AddStatusLine** není obvykle potřeba v obsluze požadavku volat, protože je interně automaticky volaná z metod **AddResponseHeader** příp. **BeginEntity**.

5.7.2.2. Metoda AddResponseHeader

Metoda **AddResponseHeader** doplní do bufferu položky záhlaví odpovědi (tj. položky, které nesouvisejí s předávanou entitou).

```
procedure THttpResponse.AddResponseHeader;
```

Parametry:

Metoda nemá žádné parametry.

Poznámky:

Interně volá metodu **AddStatusLine** (tehdy pokud nebyla zavolána) a přidá položky záhlaví **Pragma**, **Location** a **Server**.

Tuto metodu lze volat při obsluze požadavku např. tehdy, pokud odpověď neobsahuje žádná data (tj. když nevoláme metody **BeginEntity** a **EndEntity**), ale je potřeba odeslat výše zmíněné položky záhlaví.

5.7.2.3. Metoda BeginEntity

Volání metody **BeginEntity** vymezuje začátek plnění dat entity odpovědi. Do bufferu odpovědi uloží všechny potřebné položky záhlaví entity.

```
procedure THttpResponse.BeginEntity;
```

Parametry:

Metoda nemá žádné parametry.

Poznámky:

Po zavolání této metody lze zapisovat do bufferu odpovědí (tj. do položky **Content**) libovolná data. Po zápisu všech dat je nutné zavolat metodu **EndEntity**, která entitu uzavře.

Metoda **BeginEntity** interně volá metodu **AddResponseHeader** a přidává do bufferu odpovědí záhlaví entity, tj. položky **Allow**, **ContentEncoding**, **ContentType**, **Expires**, **LastModified**.

Příklad:

```
...{ kód obsluhy požadavku }
Response^.BeginEntity;
Response^.Content.WriteStr( "Text odpovedi" );
Response^.EndEntity;
...
```

5.7.2.4. Metoda EndEntity

Metoda **EndEntity** uzavře data entity. Tuto metodu je možné zavolat jen v případě, pokud byla dříve volaná metoda **BeginEntity**.

```
procedure THttpResponse.EndEntity;
```

Parametry:

Metoda nemá žádné parametry.

Poznámky:

Metoda **EndEntity** zjistí délku dat entity zapsané do bufferu odpovědi a zapíše jí do záhlaví odpovědi. Metodu **EndEntity** je nutné zavolat poté co obsluha požadavku vyplní data entity.

5.8. Třída THttpResponseText

Třída **THttpResponseText** zajišťuje buffer pro text odpovědi serveru (třídy **THttpResponse**). Data bufferu jsou alokována na heapu. Instance této třídy jsou vytvářeny a rušeny automaticky se vznikem a zánikem HTTP relace. Třída definuje několik metod pro práci s textem (**AppendXXX**, které připojují zadaný text na konec textu v bufferu). Všechny tyto metody kontrolují, zda nedochází k přepisu dat mimo buffer. Zda došlo k zaplnění bufferu lze zjišťovat pomocí metody **BufferFull**.

5.8.1. Metody třídy THttpResponseText

5.8.1.1. Metoda BufferFull

Metoda **BufferFull** zjišťuje zda došlo k úplnému zaplnění bufferu, tzn. že není možné zapsat do bufferu žádný další text.

```
function THttpResponseText.BufferFull: Boolean;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda **BufferFull** vrací hodnotu True pokud je buffer plný a False, když plný není.

Poznámky:

Metodu **BufferFull** lze volat např. po zapsání textu do bufferu (po volání jedné z metod **AppendXXX**)

5.8.1.2. Metoda BufferAvail

Metoda **BufferAvail** vrací počet znaků zbývajících do konce bufferu.

```
function THttpResponseText.BufferAvail: Word;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda **BufferAvail** vrací počet znaků zbývajících do konce bufferu. Tj. rozdíl mezi velikostí bufferu a počtem již zapsaných znaků.

5.8.1.3. Metoda Append

Metoda **Append** zapisuje libovolný text na konec textu v bufferu.

```
procedure THttpResponseText.Append( AText: PChar );
```

Parametry:

AText	Ukazatel na text zapisovaný do bufferu. Text musí být ukončen znakem #0.

Poznámky:
Pokud při zápisu dojde k zaplnění bufferu, pak je zapsáno jen tolik znaků, kolik je možné. Tento stav lze testovat pomocí metody **BufferFull**.

5.8.1.4. Metoda AppendBuff

Metoda **AppendBuff** zapisuje libovolný text na konec textu v bufferu.

```
procedure THttpResponseText.AppendBuff( ABuff: Pointer;
            ASize: Word );
```

Parametry:

ABuff	
ASize	Počet znaku v bufferu.

Poznámky:
Pokud při zápisu dojde k zaplnění bufferu, pak do zapsáno jen tolik znaků, kolik je

možné. Tento stav lze testovat pomocí metody **BufferFull**.

5.8.1.5. Metoda AppendStr

Metoda **AppendStr** zapisuje libovolný znakový řetězec na konec textu v bufferu.

```
procedure THttpResponseText.AppendStr( const AStr: string );
```

Parametry:

AStr	Zapisovaný znakový řetězec.

Poznámky:
Pokud při zápisu dojde k zaplnění bufferu, pak je zapsáno jen tolik znaků, kolik je možné. Tento stav lze testovat pomocí metody **BufferFull**.

5.8.1.6. Metoda AppendInt

Metoda **AppendInt** zapisuje číslo typu Longint na konec textu v bufferu.

```
procedure THttpResponseText.AppendInt( AValue: Longint );
```

Parametry:

AValue	Hodnota zapisovaného čísla.

Poznámky:
Pokud při zápisu dojde k zaplnění bufferu, pak je zapsáno jen tolik znaků, kolik je možné. Tento stav lze testovat pomocí metody **BufferFull**.

5.8.1.7. Metoda AppendReal

Metoda **AppendReal** zapisuje číslo typu Real na konec v bufferu.

```
procedure THttpResponseText.AppendReal( AValue: Real;
            AWidth, ADecimal: Integer );
```

Parametry:

AValue	Hodnota zapisovaného čísla.
AWidth	Celkový počet míst.
ADecimal	Počet desetinných míst.

Poznámky:
Parametry AWidth a ADecimal mají shodný význam se stejnojmennými parametry procedury Str ze standardní jednotky System. Pokud při zápisu dojde k zaplnění bufferu, pak je zapsáno jen tolik znaků, kolik je možné. Tento stav lze testovat pomocí metody **BufferFull**.

5.8.1.8. Metoda AppendTime

Metoda **AppendTime** zapisuje řetězec reprezentující čas a datum na konec textu v bufferu.

```
procedure THttpResponseText.AppendTime( const ATime: Longint );
```

Parametry:

ATime	Čas ve spakovaném tvaru (např. návratová hodnota funkce HttpGetTime nebo PackTime)

Poznámky:
Čas je do bufferu uložen ve doporučeném tvaru podle RFC-1945:

"DDD, dd mmm yyyy hh:mm:ss GMT"

DDD	je anglická zkratka pro den v týdnu (Sun, Mon, Tue, Wed, Thu, Fri, Sat)
dd	je den v měsící 00-31
mmm	je anglická zkratka pro měsíc (Jan, Feb, Mar, Apr, May, Jun, Jul, Sep, Oct, Nov, Dec)
hh	je hodina 00-23
mm	je minuta 00-59
ss	je sekunda 00-59
GMT	je text „GMT"

Příklad: Sun, 13 May 2003 15:10:32 GMT

Pokud při zápisu dojde k zaplnění bufferu, pak je zapsáno jen tolik znaků, kolik je možné. Tento stav lze testovat pomocí metody **BufferFull**.

5.9. Třída THttpQueryParser

Třída **THttpQueryParser** slouží ke zpracování parametrů dotazu na konci URL, tj. textu za znakem '?'. Tento řetězec je obvykle uložen v položce **Query** (v případě metody GET) nebo v položce **Content** (v případě metody POST) instance třídy **THttpRequest**.

Metoda **Parse** rozdělí řetězec parametrů na jednotlivé parametry, ke kterým je možno přistupovat samostatně. Parametry musí být ve standardním tvaru:
"param1=hodnota1&param2=hodnota2&…", tj. název parametru následuje znak '=' a jednotlivé parametry jsou odděleny znakem '&'. Znak '+' je nahrazen znakem ' ', a trojice %xx, kde xx je hexadecimální zápis ASCII kódu znaku je nahrazen odpovídajícím znakem. Např. řetězec:

"Jmeno=Pepa+Novak&Pozice=Reditel+%21" bude rozdělen na dva parametry:

Parametr “Jmeno” s hodnotou “Pepa Novak”
Parametr “Pozice” s hodnotou “Reditel !”

5.9.1. Položky třídy THttpQueryParser

Všechny položky třídy **THttpQueryParser** jsou určeny pouze ke čtení.

5.9.1.1. MaxCount

MaxCount : Integer;

Maximální počet parametrů, který lze zpracovat. Pokud dotaz obsahuje více parametrů než MaxCount, jsou parametry navíc ignorovány. Tuto hodnotu nastavuje konstruktor třídy - **Init**.

5.9.1.2. Count

Count : Integer

Aktuální počet zpracovaných parametrů. Zpracované parametry jsou uloženy v položce Params. Zpracování parametru provádí metoda **Parse**.

5.9.1.3. Params

Params : PAHttpQueryParam;

Pole zpracovaných parametrů. Počet zpracovaných parametrů je uložen v položce Count a pole je indexované od 0 do Count – 1 . Ostatní položky pole jsou nedefinované.

Každá položka pole je struktura **THttpQueryParam**:

```
THttpQueryParam =
record
  Name  : PChar;  { Název parametru }
  Value : PChar;  { Hodnota parametru }
end;
```

Příklad:

```
  { Vypíše všechny parametry a jejich hodnoty }
  with QueryParser do
    for I := 0 to Count - 1 do
      WriteLn( Params^[I].Name, '=', Params^[I].Value );
```

5.9.2. Metody třídy THttpQueryParser

5.9.2.1. Konstruktor Init

Konstruktor **Init** inicializuje instanci třídy **THttpQueryParser**.

```
constructor THttpQueryParser( AMaxCount: Integer );
```

Parametry:

AMaxCount Maximální počet parametrů.

Poznámky:
Kontruktor **Init** iniciazuje třídu **THttpQueryParser**, tj. alokuje na heapu pole pro parametry o velikosti 8 x AMaxCount bajtů. Položka Count je nastavena na počáteční hodnotu 0.

5.9.2.2. Destruktor Done

Destruktor **Done** uvolní položky instance třídy **THttpQueryParser**.

```
destructor THttpQueryParser.Done;
```

Parametry:

Destruktor nemá žádné parametry.

Poznámky:

5.9.2.3. Metoda Parse

Metoda **Parse** zpracovává řetězec parametrů dotazu a vytváří snadno spravovatelný seznam parametrů.

```
procedure THttpQueryParser.Parse( AQuery: PChar );
```

Parametry:

AQuery Zpracovávaný znakový řetězec s parametry dotazu zakončený znakem #0.

Poznámky:

Metoda **Parse** rozdělí řetězec parametrů na jednotlivé parametry, ke kterým je možno přistupovat samostatně. Parametry musí být ve standardním tvaru:
"param1=hodnota1&param2=hodnota2&…", tj. název parametru následuje znak '=' a jednotlivé parametry jsou odděleny znakem '&'. Znak '+' je nahrazen znakem ' ', a trojice %xx, kde xx je hexadecimální zápis ASCII kódu znaku je nahrazen odpovídajícím znakem. Např. řetězec:

"Jmeno=Pepa+Novak&Pozice=Reditel+%21" bude rozdělen na dva parametry:

Parametr "Jmeno" s hodnotou "Pepa Novak"
Paremetr "Pozice" s hodnotou "Reditel !"

Zracované parametry dotazu jsou uloženy do položky **Params**, počet zpracovaných parametrů je uložen do položky **Count**. K parametrům lze přistupovat buď přímo pomocí položky Params nebo pomocí metod **GetValueOf** a **Find**.

Předaný řetězec parametrů je po zavolání metody **Parse** modifikován. Znaky '&' a '=' jsou nahrazeny #0 a zástupné symboly %xx jsou nahrazeny odpovídajícími znaky. Do pole **Params** jsou uloženy pouze odkazy do takto zpracovaného řetězce.

Parametry dotazu se nachází buď v položce THttpRequest.Query (v případě použití metody GET) nebo v položce THttpRequest.Content (v případě použití metody POST).

5.9.2.4. Metoda Find

Metoda **Find** hledá v poli zpracovaných parametrů parametr se zadaným jménem a vrací jeho index.

```
functino THttpQueryParser.Find( AName: PChar ): Integer;
```

Parametry:

AName	Název požadovaného parametru zakončený znakem #0.

Návratové hodnoty:

V případě, že je zadaný parametr nalezen, vrací metoda index do pole Params. V opačném případě vrací hodnotu –1.

Poznámky:

Metoda **Find** hledá sekvenčním způsobem zadaný parametr. Při hledání se rozlišují malé a velké znaky.

5.9.2.5. Metoda GetValueOf

Metoda **GetValueOf** hledá v poli zpracovaných parametrů parametr se zadaným jménem a vrací jeho hodnotu.

```
function THttpQueryParser.GetValueOf( AName: PChar ): PChar;
```

Parametry:

AName	Název požadovaného parametru zakončený znakem #0.

Návratové hodnoty:

V případě, že je zadaný parametr nalezen, vrací metoda jeho hodnotu. V opačném případě vrací hodnotu **nil**.

Poznámky:

Metoda **GetValueOf** hledá sekvenčním způsobem zadaný parametr. Při hledání se rozlišují malé a velké znaky.

6. Zpracování požadavku pomocí vnořeného automatu

V některých případech není možné reagovat na požadavek klienta okamžitě. Pokud bychom v rámci obsluhy čekali na dokončení déletrvající operace (např. při vyčítání parametrů z jiného zařízení pomocí sériové linky), pak po tuto dobu nebudou obsluhovány žádné jiné požadavky klientů, přestože by server mohl reagovat okamžitě. Tuto situaci lze řešit pomocí jednoduchého stavového automatu vnořeného do obsluhy požadavku. Základní postup je naznačen v této kapitole.

Základními prostředky k implementaci automatu v obslužné rutině jsou:

- Návratová hodnota obslužné rutiny
- Ukazatel na obslužnou rutinu relace
- Položka Context

6.1. Návratová hodnota obslužné rutiny

Pro každý požadavek klienta je vytvořena vlastní instance relace, tj. instance třídy **THttpSession**. Ukazatel na vytvořenou instance je předán jako parametr funkci obsluhující požadavky klienta. Jako první je vždy zavolána funkce specifikovaná při inicializaci instance třídy **THttpServer** (položka **DataProvider**). Tato funkce má následující prototyp:

```
THttpDataProvider = function( ASession: PHttpSession ): Boolean;
```

Obslužná rutina na základě položek objektu ASession.Request zjistí požadovaný dokument a odpoví vyplněním položky ASession.Response. Poté co, je ASession.**Response** vrátí funkce hodnotu True, čímž signalizuje, že požadavek byl zpracován a již není potřeba znovu volat tuto funkci. Pokud vrátí hodnotu False, pak bude v dalším tiku (volání metody Tick třídy **THttpServer**) tato obslužná rutina zavolána znovu.

6.2. Ukazatel na obslužnou rutinu relace

Každá relace, tj. instance třídy **THttpSession** obsahuje položku **DataProvider**. Tato položka je při vytvoření relace inicializována na hodnotu stejnojmenné položky instance **THttpServer** (tj. na implicitní obslužnou rutinu). Pokud v rámci obslužné rutiny změníme hodnotu položky **DataProvider** na jinou funkcí třídy **THttpSession** a vrátíme hodnotu False, bude v příštím tiku zavolána nově specifikovaná funkce.

Zde se již rýsuje možnost vytvoření automatu pomocí posloupnosti volání různých funkcí, z nichž až poslední vrátí hodnotu True. Následující příklad schématicky ukazuje, jak lze tímto způsobem čekat na odpověď jiného vzdáleného zařízení připojeného např. po sériové lince. Příklad je pouze ilustrativní, reálná aplikace bude muset řešit navíc chybové stavy apod.

```
{ Implicitní obsluha všech požadavků specifikovaná při inicializaci
  třídy THttpServer }

function DataProvider_S0( ASession: PHttpSession ): Boolean;
var
  DocId : Integer;
  Index : Integer;
begin
  DocId := GetPathSegment( ASession^.Request.Path, @MyDocuments,
                           Index );
  case DocId of

    0: begin { Zpracování dokumentu 0 }
         { V tomto případě lze odeslat data ihned }
         with ASession^.Request do
         begin
            { Vyplnění dat odpovědi }
         end;
         DataProvider_S0 := True;
       end;

    1: begin { Zpracování dokumentu 1 }
         { V tomto případě data ihned odeslat nelze, protože je
           musíme nejprve získat z jiného zařízení }

         { První část obsluhy požadavku }
         PosliDotazPoSerioveLince;

         { Dokončení obsluhy požadavku provede funkce
           DataProvider_S1 }
         ASession^.DataProvider := DataProvider_S1;
         DataProvider_S0 := False;
       end;

  else
    { Neplatný dokument }
    DataProvider_S0 := True;
  end;
end;

{ Dokončení obsluhy požadavku na dokument 1 }

function DataProvider_S1( ASession: PHttpSession ): Boolean;
begin
  if OdpovedZeSerioveLinkyDorucena then
  begin

    with ASession^.Response do
      { zde vyplníme data odpovědi }
    end;

    DataProvider_S1 := True;
  end
  else begin
    { Odpověd zatím nepřišla }
    DataProvider_S1 := False;
  end;
end;
```

6.3. Položka Context

K udržování souvislosti jednotlivých volání obslužných rutin (tzv. kontext), složí

položka **Context** instance třídy **THttpSession**. Položka kontext je definována jako netypový ukazatel (typ Pointer). Před prvním voláním obslužné rutiny je položka **Context** inicializována na hodnotu **nil**. Při dalších voláních obslužné rutiny zůstává položka nezměněna. Do této položky může obslužná rutina uložit libovolnou hodnotu, např. číslo zprávy posílané po sériové lince, identifikátor dokumentu, stav vnořeného automatu apod.

Následující příklad je modifikací příkladu v předchozí kapitole. Do nižších 16 bitů položky **Context** se ukládá identifikátor dokumentu a funkce DataProvider_S1 dokončuje obsluhu požadavku na dokument 1 a 2 na základě její hodnoty.

```
{ Implicitní obsluha všech požadavků specifikovaná při inicializaci
  třídy THttpServer }

function DataProvider_S0( ASession: PHttpSession ): Boolean;
var
  DocId : Integer;
  Index : Integer;
begin
  DocId := GetPathSegment( ASession^.Request.Path, @MyDocuments,
                           Index );
  case DocId of

    0: begin { Zpracování dokumentu 0 }
         { V tomto případě lze odeslat data ihned }
         with ASession^.Request do
         begin
            { Vyplnění dat odpovědi }
         end;
         DataProvider_S0 := True;
       end;

    1: begin { Zpracování dokumentu 1 }
         { V tomto případě data ihned odeslat nelze, protože je
           musíme nejprve získat z jiného zařízení }

         PtrRec(Context).Lo := 1;

         { První část obsluhy požadavku }
         PosliDotaz_1_PoSerioveLince;

         { Dokončení obsluhy požadavku provede funkce
           DataProvider_S1 }
         ASession^.DataProvider := DataProvider_S1;
         DataProvider_S0 := False;
       end;

    2: begin { Zpracování dokumentu 2 }
         { V tomto případě data ihned odeslat nelze, protože je
           musíme nejprve získat z jiného zařízení }

         PtrRec(Context).Lo := 2;

         { První část obsluhy požadavku }
         PosliDotaz_2_PoSerioveLince;

         { Dokončení obsluhy požadavku provede funkce
           DataProvider_S1 }
         ASession^.DataProvider := DataProvider_S1;
         DataProvider_S0 := False;
       end;
```

```
  else
    { Neplatný dokument }
    DataProvider_S0 := True;
  end;
end;

{ Dokončení obsluhy požadavku na dokument 1 a 2 }

function DataProvider_S1( ASession: PHttpSession ): Boolean;
begin
  if OdpovedZeSerioveLinkyDorucena then
  begin

    with ASession^.Response do
     { zde vyplníme data odpovědi }

     case PtrRec(ASession^.DocId).Lo of
       1: { Dokument 1 };
       2: { Dokument 2 };
     end;

    end;

    DataProvider_S1 := True;
  end
  else begin
    { Odpověd zatím nepřišla }
    DataProvider_S1 := False;
  end;
end;
```

7. Příklad

V této kapitole je uveden ilustrativní příklad, jak vytvořit jednoduchou aplikaci, která se chová jako jednoduchý HTTP server. Tato aplikace pro zjednodušení nevyužívá jádra reálného času (knihovny Kernel).

```
program HttpSvr;

uses
  Crt,
  CoBase,
  CoEth01,
  CoINet,
  Http;


  procedure HttpErrorHandler( Session: PHttpSession;
    const AText: string ); far; forward;

  function HttpDataProvider( Session: PHttpSession ):
    Boolean; far; forward;

  function GetDocRoot( ASession: PHttpSession; APath: PChar;
    AIndex: Integer ): Boolean; far; forward;

  function GetDocPage1( ASession: PHttpSession; APath: PChar;
    AIndex: Integer ): Boolean; far; forward;


  {-------------------------------
  { HttpServer
  {---
  { Spusteni HTTP serveru,
  { procedura se ukonci po stisku klavesy
  {---
  }

  procedure HttpServer;
  var
    Status  : TCoStatus;
    HttpSvr : THttpServer;
  begin

    { Inicializace TCP/IP zasobniku }
    { Bazova adresa IOETH01 je $300 }
    { IP adresa a Maska je 192.168.1.200 a 255.255.255.0 }

    Status := NetOpenStack( 'ETH01', 'IOBASE=$300',
      'IPADDR="192.168.200" NETMASK="255.255.255.0"', '' );

    if Status <> CST_SUCCESS then
    begin
      { Chyba pri inicializaci TCP/IP zasobniku }
      HttpErrorHandler( nil, 'NetOpenStack() failed: ' +
                             CoStatusToStr( Status ) );
      Exit;
    end;

    { Inicializace instance HTTP serveru }
    { Maximalni pocet soubeznych relaci je 2, maximalni velikost
      dotazu i odpovedi je 1024B }
    HttpSvr.Init( 2, '', 1024, 1024 );
```

```
  { Funkce poskytujici data serveru }
  HttpSvr.DataProvider := HttpDataProvider;

  { Funkce reagujici na chyby serveru }
  HttpSvr.ErrorHandler := HttpErrorHandler;

  { Periodicke tikani s HTTP serverem do stisku libovolne klavesy }

  repeat
    if KeyPressed then
    begin
      ReadKey;
      HttpSvr.Stop;
    end;
  until HttpSvr.Tick;

  { Uvolneni serveru z pameti }
  HttpSvr.Done;

  { Uvolneni TCP/IP zasobniku }
  NetCloseStack( '' );

end;

{-------------------------------
{ HttpErrorHandler
{---
{ Funkce reagujici na chyby serveru
{---
{ Tuto funkci si upravte podle potreby aplikace.
}

procedure HttpErrorHandler( Session: PHttpSession;
  const AText: string );
begin
  if Session <> nil then
  begin
    WriteLn( 'Session: ', AText );
  end
  else begin
    WriteLn( 'Server: ', AText );
  end;
end;

{-------------------------------
{ HttpDataProvider
{---
{ Funkce poskytujici data serveru
{---
{ HttpDataProvider vola funkce, ktere poskytuji data
{ jednotlivych stranek
}


function HttpDataProvider( Session: PHttpSession ): Boolean;
const
  Document: array[0..2] of THttpDispatchItem =
  (
    ( Segment: '';             Dispatcher: GetDocRoot  ),
    ( Segment: 'page1.html';   Dispatcher: GetDocPage1 ),
    ( Segment: nil;            Dispatcher: nil )
  );
begin
```

```
  with Session^.Request do
  begin
    { Podle typu metody rozhodneme, kterou tabulku pouzijeme }

    case Method of

      mtdGET:
      begin
        HttpDataProvider := HttpDispatchSegment(
          Session^.Request.Path, @Document, Session );
      end;

    else
      { Nepodporovana metoda (jina nez GET) }
      with Session^.Response do
      begin
        StatusCode := HTE_METHOD_NOT_ALLOWED;
        { Odpoved je pripravena, nechame ji odeslat a tedy
          vratime True }
        HttpDataProvider := True;
      end;
    end;
  end;

end;


{-------------------------------
{ GetDocRoot
{---
{ Vyplneni odpovedi na dotaz GET /
{---
}

function GetDocRoot( ASession: PHttpSession; APath: PChar;
  AIndex: Integer ): Boolean;
begin
  with ASession^.Response do
  begin
    StatusCode := HTE_OK;

    BeginEntity;
    with Content do
    begin
      Append( '<HTML>' );
      Append(   '<BODY>' );
      Append(     '<B>WWW Server - / (root)</B>' );
      Append(     '<P>Odkaz na stranku <A ' +
                   'HREF="/page1.html">page1.html</A>' );
      Append(   '</BODY>' );
      Append( '</HTML>' );
    end;

    EndEntity;
  end;

  GetDocRoot := True;
end;


{-------------------------------
{ GetDocPage1
{---
{ Vyplneni odpovedi na dotaz GET /page1.html
{---
```

```
  }

  function GetDocPage1( ASession: PHttpSession; APath: PChar;
    AIndex: Integer ): Boolean;
  begin
    with ASession^.Response do
    begin
      StatusCode := HTE_OK;

      BeginEntity;
      with Content do
      begin
        Append( '<HTML>' );
        Append(   '<BODY>' );
        Append(     '<B>WWW Server - page1.html</B>' );
        Append(   '</BODY>' );
        Append( '</HTML>' );
      end;

      EndEntity;
    end;

    GetDocPage1 := True;
  end;


{ Telo hlavniho programu }

begin
  HttpServer;
end.
```