HttpSvr User's Guide

March 2015

Version 2.0

TICSOFTWARE.COM

The Leader in NonStop Modernization

Proprietary & Confidential

© 2021 Transaction Innovation Corporation

All rights reserved

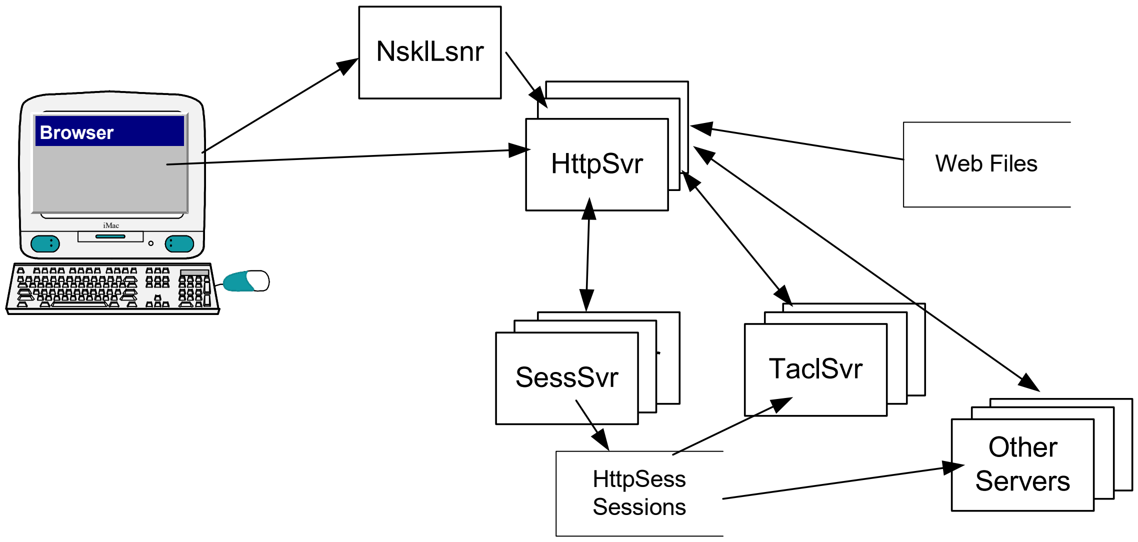
Table of Contents

Version History

Version	Date	Description/
1.0	2009/03/23	Initial Draft
1.5	2010/05/04	Added error handling and trace features
1.6	2012/09/12	Minor bug fixes
1.9	2013/07/01	Additional XML configuration parameters
2.0	2015/02/22	Changes in IPM parameters

1 Overview

HttpSvr is a NonStop™ Guardian based application providing some webserver functionality. TAS’s primary utility is access to Pathway™ serverclasses. A request/reply IPC pair is defined for servers wanting to communicate with web browsers. Session management is provided. A server to run TACL macros is also included. Pair of sample servers, written in C++, is included.



2 Installation

Installation of TAS is done using the NSKInstaller.exe on the PC. A help file for NSKInstaller.exe is included. There are two text files that should be customized for each installation. Directories.txt contains the directory information for TAS. Servers.txt contains the server information for TAS.

As a final step, a TACL macro, HTMACRO, is run to build all of the needed configuration files. These EDIT files can be modified based on the following documentation.

When HTMACRO runs, it will replace the tcpip address and port in all the files in the webfiles subvolume with the correct values. You might have to correct those in some cases depending on the network configurations.

Note: NsklLsnr and TAS will NOT stop when the pathmon is shutdown and must be explicitly stopped via a TACL stop.

3 Configuration

3.1 *Pathway*

There are several Pathway configuration files created by the installer.

Pathcnfg	This file is used by Pathcold to initialize the Pathway environment. You might want to edit this file to change CPUs and other configuration values.
Pathcold	Initializes the Pathway environment. You must run this first.
Pathcool	Cool starts the Pathway environment.

3.2 *NsklLsnr*

NsklLsnr is the program that listens to the TCP/IP port for browser connects. It reads its configuration values from NSKLCNFG. The only value that might need to be changed in NSKLCNFG is the port value in the [nskllsnr] section.

port = nnn

3.3 *TAS*

The TAS communicates with browser. It receives requests, determines if just a file is to be returned or if the request is to be passed to a server. TAS also detects replacement variables in the data to be returned to the browser and sends the replacement request to SessSvr. The primary use of this feature is obtaining session ids. See below for more information about variable replacement.

TAS gets a message from NSKLLsnr about a TCPIP connection request and receives the data from the socket. TAS does all I/O nowaited so it can timeout if needed if a server is non-responsive.

<HttpSvr>

<License Key="xxxx"/>	The license key for TAS.
<Log_Level Level="Verbose | Errors"	Verbose will cause extra detail to be written to the OUT file. To get the most detail, include PARAM DEBUGFLAG TRUE when starting TAS .
/>	
<TCPIP	
TCPIP_Process="xxx"	The process name of the TCPIP process to use. Supersedes the ASSIGN.
Http_Server_Name="xxx"	The name to put in the http response header for this NSK host. Default is "TicHttpSvr".
/>	
<Pathway	
My_Pathmon="xxx"	The process name of the Pathmon managing this serverclass. Supersedes the PARAM.
NSKLLsnr_Serverclass="xxx"	The serverclass name for NSKLLSNR. Supersedes the PARAM. The default is ANSKLLSNR. NSKLLSNR must start before the HTTPSVR serverclass does.
My_Serverclass="HTTPSVR"	The name of our serverclass. It is used for restarts by NSKLLSNR.
Restart_Timer="n"	How long, after detecting that NSKLLSNR has died, before HTTPSVR will attempt to restart it.

```
      My_Pathway_Only=”Yes | No”If Yes, only serverclass in the My_Pathmon will be
                                allowed.
      />
  <Servers>                     A list of servers that can be talked to.  There will be none
                                or more configured.

      <Server
        Alias=”xxx”             The name used in the GET or POST request for this
                                server.
        Pathmon=”xxx”           The pathmon managing this server.
        Serverclass=”xxx”       The serverclass for this server.
        Timeout=”n”             How long to wait for this server to respond.
      />
  </Servers>                    End of Servers list
  <Directories>                 A list of directories to use
      <Server_Directory
        Alias=”xxx” />          The name to use to indicate it is a server, not a file.  The
                                default is “svr”.
      <Default_Directory
        Subvol=”xxx”            The subvol to use for requests with no alias provided.
        Default_File=”xxx”      The filename of a file in the default subvol to return is no
                                filename was given in the request.
        />
      <Default_Directory
        Alias=”xxx”             The name to use indicating it is this subvol to be used.
        Subvol=”xxx”            The subvol to use for requests with this alias
        Default_File=”xxx”      The filename of a file in the subvol to return is no filename
                                was given in the request.
        />
      <Temp_Files
        Subvol=”xxx”            The subvol to use for temporary files exchanged between
                                httpsvr and other servers.
        />
  </Directories>                End of directories list
  <Replacment_Variables
      Pathmon=”xxx”             The pathmon managing the SessSvr serverclass.
      Serverclass=”xxx”         The serverclass of the SessSvr.
      Tag=”x” />                The character indicating a start of a replacement variable.
                                The default is “~”.  See below for more information on
                                replacement variables.
</HttpSvr>                    End of HttpSvr
```

3.3.1 Variable replacement in response data:

When TAS is returning data to the browser, it will scan the data for replacement variables. The Replacment_Variables Tag character is used. The default is “~”. Each variable to be replaced must be preceded by two (2) Tag characters and terminated by the Tag character. The only supported variable at this time is ~~session~. The variable name is sent to the SessSvr and the value returned is inserted in the data.

3.4 SessSvr

The SessSvr creates and logs sessions for use by other servers. It also ages the records and deletes any that have expired.

```
<SessSvr>
  <Log_Level  Level="Verbose | Errors"
                                      Verbose will cause extra detail to be written to the OUT
                                      file.  To get the most detail, include PARAM
                                      DEBUGFLAG TRUE when starting SessSvr.
  />
  <Session
      Filename="xxx"                 The filename of the session file.  Default is "HTTPSESS";
      KeepMinutes="n"                How long a session is valid after its last request is
                                     received.  The default is 20.
      />
</SessSvr>                          End of SessSvr
```

3.5 TaclSvr

When TaclSvr starts, it reads an edit CONFIG file with XML configuration information telling it what to process. Within the configuration file given in ASSIGN CONFIG, you can include, #include and #includeassign statements to source in XML from other files.

How TaclSvr works is defined in "Using TaclSvr".

```
<TaclSvr>
  <Log_Level  Level="Verbose | Errors"
                                      Verbose will cause extra detail to be written to the OUT
                                      file.  To get the most detail, include PARAM
                                      DEBUGFLAG TRUE when starting TaclSvr.

      Filename="fname"                The status file that TaclSvr will log requests to.  The
                                      default is TaclSLog.

      KeepDays="n"                    How many days worth of status to keep.  Records older
                                      than that are deleted.

      CleanHour="n"                   When to clean up the status records.

  />
 <Report_Only_N_Consecutive_Errors
    Value="n"/>                      If the same error occurs repeatedly before a success, how
                                     many to report.  Default is 3.
 <TACL_Info
    Program="xxx"                    The location (fully qualified) of the program, usually TACL,
                                     to run.  Default is "$SYSTEM.SYSTEM.TACL".
    Default_Subvol="xxx"             The subvol to volume to after doing a logon.  This is the
                                     subvolume where the TACL macros reside.
    Initial_Macro="xxx"              Optional TACL macro to run after moving to the default
                                     subvol.
    Timeout="n"                      How many seconds to wait for Macros to complete.
    Log_All_Output="Yes | No"        If Yes, all output received from TACL is echoed to the
                                     TaclSvr OUT file.
```

Security_ID=”xxx”	Default security id to use if a configured macro does not have one of its own but requires security.
/>	Required end of <TACL_Info
<TACL_Tags	"Tags" or text checked at the beginning of each line received from TACL to see if it is to be handled.
Error="xxxx"	Default is "TACLSVR_Error". The text on this line, following the tag will be returned as the ErrorText for HttpSvr. Processing of the TACL output is terminated upon receipt of this tag.
Success="xxx"	Default is "TACLSVR_Success". Signals successful end of the macro's execution.
Macro_Not_Found="xxx"	Default is "Expecting the name of an existing file". Used to detect the given execute macro is not found.
/>	Required end of <TACL_Tags
<TACL_Macros>	A list of macros that can be run.
<Macro	For each macro that can be run.
Name=”xxx”	The macro name. It is the name of the EDIT file TACL macro or routine to can be run.
IsSecure=”Yes | No”	If Yes, the macro request must include id=xxx where xxx is the security_id. If this macro does not have a security_id configured, the Tacl_Info Security_ID will be used. The default is “No”.
Security_ID=”xxx”	Optional security id to use for this macro.
NeedsSession=”Yes | No”	If Yes, a valid session id must be included in the request.
/>	End of each Macro definition
</TACL_Macros>	End of Tacl_Macros
<Session	
Filename=”xxx”	The filename of the session file. Default is “HTTPSESS”;
KeepMinutes=”n”	How long a session is valid after its last request is received. The default is 20.
/>	
</TaclSvr>	End of TaclSvr

3.6 Common ASSIGN and PARAM Values

ASSIGN CONFIG, <filename>	The filename where the XML configuration information is. The default values are HttpSvr – HttpCnfg TaclSvr – TaclCnfg SessSvr - SessCnfg
PARAM LOGPREFIX [NONE | TIME]	How much of the prefix to include. Default is TIME.

Examples:

Time	2005-05-19 16:53: HttpSvr - Copyright 2006 TIC
None	HttpSvr - Copyright 2006 TIC

PARAM DEBUGFLAG [TRUE | FALSE]	will turn on detailed OUT logging for everything.
PARAM ECHOCONFIG [YES | NO]	will echo, to the OUT file, the configuration values as they are processed.

HttpSvr and NsklLsnr have additional assigns and params.

ASSIGN PNAME-FILE, <filename>	The filename to hold process information for the HttpSvr that NsklLsnr needs. The default is PNAMES.
ASSIGN TCIPIP-PROCESS, <pname>	The process name of the TCPIP process to use. The default is $ZTC0.
PARAM MY-PATHMON [pname]	The process name of the Pathmon managing the servers.
PARAM PWSVRLNK-SERVERCLASS [sname]	The serverclass name of the HTTPSVRs. Needed by NSKLLSNR.
PARAM NSKLLSNR-SERVERCLASS [sname]	The serverclass name of the NSKLLSNR. Needed by HTTPSVR.

3.7 Example XML Configuration

```
<HttpSvr>
  <Log_Level
     Level="Verbose" />
  <License_Keys
     Httpsvr="HTTPSVR-xxxx" />
  <TCPIP
    TCPIP_Process="$ztc0"
    Http_Server_Name="TicK2k"/>
  <Pathway
    My_Pathmon="$HTTP"
    NSKLLsnr_Serverclass="ANSKLLSNR"
    My_Serverclass="HTTPSvr"
    />
  <Directories>
    <Server_Directory
       Alias="Svr"
       />
    <Default_Directory
       Subvol="$sysd30.donhttpd"
       Default_File="Default"
       />
    <Temp_Files
       Subvol="$sysd30.donhttpt"
       />
    <Directory
       Subvol="$sysd30.donhttp"
       Alias="Test"
       Default_File="Default"
  />
  </Directories>
  <Servers>
    <Server
       Alias="Echo"
       Pathmon="$HTTP"
       Serverclass="EchoSvr"
       Timeout="120"
      />
    <Server
       Alias="File"
       Pathmon="$HTTP"
       Serverclass="FileSvr"
       Timeout="120"
      />
    <Server
       Alias="Tacl"
       Pathmon="$HTTP"
       Serverclass="TaclSvr"
       Timeout="120"
      />
  </Servers>
</HttpSvr>
```

4 Using XML

The following is the XML syntax that is recognized.

```
Expecting the following
        <name>
        </name>
        <name attr="val"/>  (1 or more attr/val pairs)
        <name attr="val">       (1 or more attr/val pairs,
                                        more attr="val" pairs in next <> or
                                        /name (ending feature))

        afetr <name can be
            >
            />
            attr="val"
        after attr="val" can be
            >
            />
            attr="val"
        after </name can be
            >
    Whitespace handling:
        Cannot be spaces between < and name.
        Cannot be spaces between / and >
        Cr and/or Lf are treated as a single space
        Spaces allowed before and after =
        Spaces allowed before > or />
```

To include a double quote (") in an attribute value, use paired "". As an example, if you wanted to end up with quotes around a value you would enter """Joe Smoo""". The result would be "Joe Smoo". Note this is a deviation from the XML standard. Attribute values are not fully normalized according to the XML standard. Any characters can be in the value including multiple spaces.
Comments begin with <!-- and end with -->.
Comments can contain any data except the literal string -->.

```
    The following is what is allowed/expected after each:
        Entity (may have attributes)
            Entity
            Entity-end
```

If Entity ends with />, there will be no subenties for it
If Entity is an array, each time another occurrence of Entity is encountered it will another branch will be added for this entity.

Entity-end must be </name>

5 Using TaclSvr

TaclSvr will parse the httpData (or the contents of the file passed from HttpSvr). The format of that data will be
Var=val&var=val&var=val…

If the macro requires a session, the first pair must be sesson=xxxx where xxxx is the session created by the SessSvr.

The next pair (or first pair if session isn't required) must be macro=xxxx where xxxx is the macro to run.

If the macro IsSecure, the next pair must be id=xxxx where xxxx is either the security_id for the macro or the default security_id.

The next pair must be numvars=n where n is how many more pairs follow. Each pair will be written to a file for the Tacl to load with each var prefixed with "v_".

As an example for the following request:

http://12.103.96.173:1620/svr/tacl?session=HTC0212020809698047470&macro=TTest&id=TLock&numvars=2&abc=today&def=noway

The variables set for the Tacl will look like

```
?Tacl Macro
#PUSH httpReplyFile
#SET httpReplyFile ZHTC002
#PUSH httpSession
#SET httpSession HTC0212020809698047470
#PUSH v_abc
#SET v_abc today
#PUSH v_def
#SET v_def noway
```

httpReplyFile will contain the filename to put the html data to return to the browser. httpSession has the session id, if any.

There is a Tacl macro that is run each time Tacl is started. It loads a routine to safely extract the variables and values even if the variable is missing.

HttpVarGet <var> <default>

Expands to the contents of var or the default if var doesn't exist.

Note that you must load the httpReplyFile with the html and then, to end, do

#output TaclSvr_Success

Example macro

```
?Tacl Macro
#FRAME
#PUSH out vfi vprocess
#SET vprocess [httpvarget v_vppd ]
[#IF [#EMPTYV vprocess] |THEN|
  #SET vfi Missing var name 'vppd'
 |ELSE|
  [#IF [#PROCESSEXISTS [vprocess]] |THEN|
    #SET vfi Process [vprocess] is running
   |ELSE|
    #SET vfi Process [vprocess] is NOT running
  ]
]
[#SET out
<html> <head> <title>TaclSvr default!</title> </head>
  <body> [vfi] </body> </html>
]
[#IF [#FILEINFO/EXISTENCE/[httpReplyFile]] |THEN|
 sink [#PURGE [httpReplyFile]]
]
vartofile out [httpReplyFile]
#OUTPUT TaclSvr_Success
#UNFRAME
```

6 Sending to Pathway Serverclasses from TACL Macros

TACL only provides WRITEREAD communication to servers which causes open and closes of those servers. This can cause the servers to stop. WRITEREADs also require a named process. The HttpSvr system includes an intermediate server to allow TACL macros to do Serverclass_Send to other Pathway servers.

To utilize these extensions, add the following line near the beginning of the macro:

sink [#load/keep 1/tpthsend]

TPTHSEND provides three routines.

HTTP_PATHSEND <pathmon> <svrcls> <to> <errstrt> <req> <reply>

<pathmon>	The Pathmon process name managing the serverclass. Example $PTH1
<svrcls>	The serverclass to send to. Example EXAMPLE-SVR.
<to>	The timeout seconds. Example 30.
<errstrt>	A large number > any replycode that the server might return. Used to piggy-back errors from the PSENDSVR back to the macro.
<req>	A TACL Struct of the request. Usually loaded from output of DDL.
<reply>	A TACL Struct of the reply. Usually loaded from output of DDL.

CONVERT_TO_PIC9 <nin> <decimalpt> <width>

<nin>	The number to convert to a COBOL PIC 9 format.
<decimalpt>	The number of digits after the implied decimal point in the COBOL number. Example. 2 for a PIC 999V99. Use 0 if no decimal point.
<width>	Total width of the resulting field.

CONVERT_FROM_PIC9 <nin> <decimalpt>

<nin>	The number to convert from a COBOL PIC 9 format.
<decimalpt>	Where the implied decimal point is in <nin>. Use 0 if no decimal point.

Example that is included.

```
?Tacl Macro
#FRAME
#PUSH out vfi vnum
#SET vnum [httpvarget v_vnum ]
#push  i cn rp
[#IF [#EMPTYV vnum] |THEN|
  #SET vfi Missing var name 'vnum'
 |ELSE|
    #SET vfi OK
]
sink [#load/keep 1/sogtacl]
sink [#load/keep 1/tpthsend]



[#SET out
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
        <TITLE>Read Contacts</TITLE>
        <LINK REL="STYLESHEET" HREF="../maincss.css">


</HEAD>

<BODY TOPMARGIN="0" LEFTMARGIN="0" MARGINHEIGHT="0" MARGINWIDTH="0">
<TABLE WIDTH=630 CELLPADDING=0 CELLSPACING=0 BORDER=0>
<TR BGCOLOR="#330066" >
        <TD WIDTH=5 BGCOLOR="#330066">&nbsp;</TD>
        <TD WIDTH=610 VALIGN=TOP STYLE="color: #99CCCC;"><B><FONT
SIZE=1>Example ASP Application</FONT></B></TD>
        <TD WIDTH=15 ALIGN=RIGHT><IMG SRC="../tcgif.gif" WIDTH=15
BORDER="0"></TD>
</TR>
</TABLE>
<TABLE WIDTH=650 CELLPADDING=0 CELLSPACING=0 BORDER=0>
<TR>
        <TD WIDTH=5 BGCOLOR="#330066">&nbsp;</TD>
        <TD BGCOLOR="#330066" VALIGN=TOP WIDTH=160>
                <BR>
                <TABLE BORDER="1" CELLPADDING="2" ALIGN="left"
BGCOLOR="#9999CC" WIDTH=150>
                <TR>
                        <TD>
                                <B>
                                <LI><A HREF="ReadC">Read Contacts</A><BR>
                                <LI><A HREF="AddC">Add Contact</A><BR>
                                <LI><A HREF="DeleteC">Delete Contact</A><BR>
                                </B>
                        </TD>
                </TR>
                </TABLE>
        </TD>
        <TD>
                <IMG SRC="../curvejpg.jpg" WIDTH=39 HEIGHT=35 BORDER=0 ALT="">
                <P><BR>
] ====     it goes here

 #SET READ^CONTACT^REQ:REQUEST^HDR:REQUEST^TYPE 902
 #SET READ^CONTACT^REQ:start^contact^number [CONVERT_TO_PIC9 [v_vstart] 0 6]
 #SET READ^CONTACT^REQ:num^wanted           [v_vnum]

 #PUSH status
 #SET status [HTTP_PATHSEND $HTTP EXAMPLE-SERVER 30 10000 READ^CONTACT^REQ
READ^CONTACT^REPLY]
 [#IF NOT [#EMPTYV status] |THEN|
    #APPEND OUT [status]
  |ELSE|

   [#IF (READ^CONTACT^REPLY:reply^hdr:reply^code <> 0) |THEN|
      #SET rp READ^CONTACT^REPLY:reply^hdr
      #APPEND OUT Error [[rp]:reply^code]:[[rp]:file^error]
    |ELSE|
      #SET i 0
      [#LOOP |WHILE| I < read^contact^reply:NUM^RETURNED |DO|
        #APPEND OUT &nbsp<BR><table rules=cols bgcolor=#99CC99
                #APPEND OUT cellpadding=2 cellspacing=2 width=450>
        #APPEND OUT <tr><td width=100><b>Contact Number:</b> </td>
                #SET rp read^contact^reply:contact^data([i]):contact^info
                #SET cn [CONVERT_FROM_PIC9 [ [rp]:contact^number ] 0]
        #APPEND OUT <td>  [cn] </td></tr>
        #APPEND OUT <tr><td><b>Last Name:</b> </td>
        #APPEND OUT <td>  [ [rp]:owner^name:last^name ] </td></tr>
        #APPEND OUT <tr><td><b>First Name:</b> </td>
        #APPEND OUT <td>  [ [rp]:owner^name:first^name ] </td></tr>
        #APPEND OUT <tr><td><b>Middle Initial:</b> </td>
        #APPEND OUT <td>  [ [rp]:owner^name:middle^initial ] </td></tr>
        #APPEND OUT <tr><td><b>Street:</b> </td>
        #APPEND OUT <td>  [ [rp]:address:street ] </td></tr>
        #APPEND OUT <tr><td><b>City:</b> </td>
        #APPEND OUT <td>  [ [rp]:address:city ] </td></tr>
        #APPEND OUT <tr><td><b>State:</b> </td>
        #APPEND OUT <td>  [ [rp]:address:state ] </td></tr>
        #APPEND OUT <tr><td><b>Zip:</b> </td>
        #APPEND OUT <td>  [ [rp]:address:zip ] </td></tr>
        #APPEND OUT <tr><td><b>Area Code:</b> </td>
        #APPEND OUT <td>  [ [rp]:PHONE^PARTS:area^code ] </td></tr>
        #APPEND OUT <tr><td><b>Local Phone:</b> </td>
        #APPEND OUT <td width=225>  [ [rp]:PHONE^PARTS:local^phone ]
</td></tr></table>

        ==#OUTPUT [CONVERT_FROM_PIC9 [cn] 0]
        #SET i [#COMPUTE i + 1]
      ]
   ]
 ]

[#APPEND OUT

        </TD>
</TR>
<TR BGCOLOR="#330066">
        <TD WIDTH=5 BGCOLOR="#330066">&nbsp;</TD>
        <TD WIDTH=160 ALIGN=RIGHT><IMG SRC="../tcgif.gif" WIDTH=15
BORDER="0"></TD>
        <TD BGCOLOR="#FFFFFF">&nbsp;</TD>
</TR>
</TABLE>
</BODY>
</HTML>
]
[#IF [#FILEINFO/EXISTENCE/[httpReplyFile]] |THEN|
 sink [#PURGE [httpReplyFile]]
]
vartofile out [httpReplyFile]
#OUTPUT TaclSvr_Mime_Type text/html
#OUTPUT TaclSvr_Success
```

7 Writing HttpSvr Server Programs

HttpSvr will send the following request to a configured server.

```
#define MAX_HTTP_DATA 29000
typedef struct HttpSvrDataRequest
{
  short      requestCode;           // will be 101
  char       ident[2];              // will be HS
  short      version;               // will be 1
    char     dataInIPM;             // will be 'Y' if data in this
                                    request.  Otherwise, in file
    char     requestType;           // G = get, P = post, O = other
  int        dataLen;
  char       dataFile[40];          // filename if data NOT in this
                                    request
  char       httpData[MAX_HTTP_DATA];
} HttpSvrDataRequest_def;
```

The contents of the httpData is all the data after the "?". If it won't fit in httpData, it will be written to a file and that filename will be put into the request.

The server must reply with

```
typedef struct HttpSvrDataReplyHeader
{
    short    replyCode;
    short    error;
    short    errorDetail;
    char     errorText[HTTP_ERRORTEXT_SIZE];
} HttpSvrDataReplyHeader_def;
```

```
typedef struct HttpSvrDataReply
{
   HttpSvrDataReplyHeader_def replyHdr;
   char     dataInIPM;             // will be 'Y' if data in this
                                   request.  Otherwise, in file
   char     purgeDataFile;         // will be 'Y' if dataFile to be
                                   purged
   int      dataLen;
   char     dataFile[40];          // filename if data NOT in this
                                   request
   char     requestDataFile[40];   // filename of request data, if
                                   any to be purged
   char     mimeType[40];
   char     httpData[MAX_HTTP_DATA];
} HttpSvrDataReply_def;
```

The data to return, either in the file or the httpData, must be the entire page or item. The http header will be constructed by HttpSvr.