53

TCP/SNA update

March 2004

In this issue

TCP/SNA Update

Published by
Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690
Fax: 214-341-7081

Editor
Fiona Hewitt
E-mail: fionah@xephon.com

Publisher
Nicole Thomas
E-mail: nicole@xephon.com

Subscriptions and back-issues

A year's subscription to *TCP/SNA Update*, comprising four quarterly issues, costs $190.00 in the USA and Canada; £130.00 in the UK; £136.00 in Europe; £142.00 in Australasia and Japan; and £140.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the March 2000 issue, are available separately to subscribers for $49.50 (£33.00) each including postage.

Editorial panel

Disclaimer

Contributions

When Xephon is given copyright, articles published in *TCP/SNA Update* are paid for at the rate of $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

TCP/SNA Update on-line

Code from *TCP/SNA Update*, and complete issues in Acrobat PDF format, can be downloaded from http://www.xephon.com/tcpsna; you will need to supply a word from the printed issue..

Dynamically invoking ftp

In today's cyber world, transferring files between systems is a very common requirement, and OS/390 (z/OS) systems cannot be excluded from this activity. VTAM and SNA on MVS have been augmented by TCP/IP on later generation MVS/ESA systems and its successors OS/390 and z/OS. TCP/IP has become an integral networking component in the mainframe world.

The common TCP/IP applications are all available on z/OS. This includes ftp (file transfer protocol), the common application for transferring files between different physical or logical systems. ftp on z/OS can be invoked interactively through TSO or through a standard z/OS batch job (see Chapters 3 and 4 in the *IP User's Guide* from the IBM Communications Server bookshelf), but you may sometimes need to perform file transfer operations within the framework of program logic.

The FTPLINK program presented with this article provides sample code to invoke ftp dynamically from within a wrapper program. This can be beneficial if you need to perform a file transfer operation that's a requirement of a larger process that involves programming decisions before and/or after the file transfer operation. Using dynamic linkage from a program environment can provide more flexibility than using a multi-step batch job with condition code checking. FTPLINK demonstrates how this can be accomplished.

On completion of the ftp operation, FTPLINK will examine the contents of the ftp OUTPUT dataset (for FTPLINK, the dataset assigned to the OUTPUT DD statement must be a real dataset on DASD, not a JES SYSOUT dataset) and determine, to the best of its abilities, whether or not the file transfer operation completed successfully. Depending on your requirements, you can include additional logic based on file transfer success or failure.

FTPLINK subscribes to all the traditional ftp requirements and inputs. The EXEC statement PARM can be used to pass parameter values to ftp – one of the more useful ones being the (EXIT parameter that indicates to ftp that it should pass back a failure return code if the ftp operation is unsuccessful. The NETRC DD statement can be used to pass netrc information to ftp. The INPUT DD statement can be used to pass ftp commands. As mentioned earlier, the OUTPUT DD is used to capture output from ftp – the output contained in this dataset is used by FTPLINK to assess the success of the file transfer operation.

Shown below is some sample JCL to linkedit FTPLINK:

```
//IEWL   EXEC PGM=HEWLHØ96,PARM='XREF,LIST,MAP'
//SYSPRINT DD  SYSOUT=*
//SYSUT1  DD  UNIT=SYSDA,SPACE=(CYL,(2,1))
//OBJECT  DD  DSN=object.library,DISP=SHR
//SYSLMOD DD  DSN=load.library,DISP=SHR
//SYSLIN  DD  *
  INCLUDE OBJECT(FTPLINK)
  ENTRY  FTPLINK
  NAME   FTPLINK(R)
```

Sample JCL for running the FTPLINK program is shown below, and also in the comments for the FTPLINK program.

```
//FTPLINK EXEC PGM=FTPLINK,REGION=8192K
//STEPLIB DD DSN=load.library,DISP=SHR
//SYSPRINT DD SYSOUT=*
//OUTPUT  DD DSN=output.dataset,DISP=SHR
//INPUT   DD DSN=ftp.input.commands,DISP=SHR
//SYSTCPD DD DSN=TCPIP.TCPDATA,DISP=SHR  <=== optional
//NETRC   DD DSN=userid.NETRC,DISP=SHR   <=== optional
```

As mentioned earlier, FTPLINK uses standard FTP inputs and outputs. This includes parameter values specified in the PARM keyword of the EXEC statement and the use of INPUT, OUTPUT, NETRC, and SYSPRINT DD statements. The *IP User’s Guide* provides details.

In order to successfully use FTPLINK, you’ll require proper security access. This will include security product OMVS segment definitions for the batch job’s userid (and its default group) and appropriate access to the IP stack.

FTPLINK will issue various messages to the operator console and the job's message log depending on the status of the file transfer operation.

Try FTPLINK in your environment. You probably have some very practical uses for a wrapper similar to it.

FTPLINK

```
*---------------------------------------------------------------*
*                                                               *
*  The FTPLINK program provides an example of dynamically invoking  *
*  the FTP TCP/IP application from within an existing program    *
*  framework.  This can prove to be extremely beneficial if a file *
*  transfer operation is integral to a sequence of events that   *
*   are happening within a program's logic.                     *
*                                                               *
*  The FTPLINK program will dynamically link to FTP and then     *
*  examine the resulting OUTPUT file contents to determine whether *
*  the FTP request was a success or not.  This check can then be *
*   incorporated into further program logic.                    *
*                                                               *
*  The dynamic invocation of FTP will subscribe to the standard  *
*  requirements of an online or direct batch invovation of FTP.  *
*  That is, an INPUT file of FTP commands should be allocated and *
*   a NETRC DD can be used as well.                             *
*                                                               *
*  The OUTPUT DD should specify a dasd dataset and not a JES     *
*  SYSOUT dataset as FTPLINK is designed to read the OUTPUT      *
*  dataset's content, as produced by FTP, from a dasd dataset.   *
*                                                               *
*  Sample JCL for invoking FTPLINK would look something like:    *
*                                                               *
*   //FTPLINK  EXEC PGM=FTPLINK,REGION=8192K                    *
*   //STEPLIB  DD   DSN=load.library,DISP=SHR                   *
*   //SYSPRINT DD   SYSOUT=*                                    *
*   //OUTPUT   DD   DSN=output.dataset,DISP=SHR                 *
*   //INPUT    DD   DSN=ftp.input.commands,DISP=SHR             *
*  //SYSTCPD DD  DSN=TCPIP.TCPDATA,DISP=SHR   <=== optional     *
*  //NETRC   DD  DSN=userid.NETRC,DISP=SHR    <=== optional     *
*                                                               *
*  The FTPLINK program will pass the parameter address it detects *
*  to the FTP application so you can specify normal FTP parameters *
*  using PARM= on the EXEC statement.  Specifying PARM='(EXIT'   *
*  will cause the FTP application to return a non-zero return code *
*  if a failure is detected in FTP.  A WTO message will be issued *
*  by the FTPLINK program in this case.  The WTO message will    *
*  indicate the FTP subcommand return code and FTPD reply code.  *
*                                                                *
*----------------------------------------------------------------*
FTPLINK CSECT
FTPLINK AMODE 31
FTPLINK RMODE 24
      STM  R14,R12,12(R13)    Save incoming registers
       LR   R12,R15           Copy module base address
      USING FTPLINK,R12       Set module addressability
       LR   R3,R13            Save incoming savearea address
       LR   R1Ø,R1            Save incoming parm address
     STORAGE OBTAIN,LENGTH=WORKLEN,LOC=BELOW
       LR   RØ,R1             Copy storage address
       LR   R13,R1            Again
       LR   R14,R1            And again
       L    R1,=A(WORKLEN)    Get storage length
       XR   R15,R15           Set fill byte value
       MVCL RØ,R14            Clear working storage
      USING WORKAREA,R13      Set addressability to workarea
       ST   R3,SAVEAREA+4     Save old savearea address
       ST   R1Ø,PARMADDR      Save incoming parm address
*----------------------------------------------------------------*
*  Any pre-FTP logic you might need could be inserted here.      *
*----------------------------------------------------------------*
       L    R1,PARMADDR       Use passed parm address for FTP
      MVC  LINKWRK(LINKLN),LINKLS Move in LINK model
      LINK EP=FTP,SF=(E,LINKWRK) Link to FTP
       ST   R15,FTPRC         Save the FTP return code
*----------------------------------------------------------------*
**     WTO  'FTP has completed'
*----------------------------------------------------------------*
      MVC  DCB1WRK(DCB1LN),OUTPUT Copy the DCB model
       OI   OPENLST,X'8Ø'     Set parm bit on
     OPEN (DCB1WRK,INPUT),MODE=31,MF=(E,OPENLST) Open the dataset
      TM   DCB1WRK+48,X'1Ø'   Open successful?
       BO   OPENOK            Yes - go on
      WTO  'FTPLINK - Open failed for OUTPUT'
       B    RETURN12          Pack it in
*----------------------------------------------------------------*
OPENOK  DS   ØH
**     WTO  'OUTPUT file opened'
*----------------------------------------------------------------*
*  The FTP request has completed and the OUTPUT dataset has been  *
*  opened successfully.  Check the data in the OUTPUT dataset to  *
*  determine if the request completed as expected.                *
*----------------------------------------------------------------*
GETLOOP DS   ØH
       GET  DCB1WRK,INBUFF    Read a record
      CLC  INBUFF(22),=C'226 Transfer complete.' A success msg?
       BE   GOODFTP           Yes - set success flag
      CLC  INBUFF(36),=C'25Ø Transfer completed successfully.' OK?
```

```
         BE    GOODFTP            Yes - set success flag
         CLC   INBUFF(9),=C'EZA1617I ' Byte count message?
         BE    GTBYTCNT           Yes - save byte count
         B     GETLOOP            Get next record
GOODFTP  DS    ØH
         OI    FLAG1,FTPOK        Set success flag
         B     GETLOOP            Get next record
GTBYTCNT DS    ØH
         LA    R1,INBUFF+9        Get starting address
         LA    R14,BYTECNT        Get target area address
         XR    R15,R15            Clear counter
         MVI   BYTECNT,C' '       Set fill byte
         MVC   BYTECNT+1(39),BYTECNT Clear the target area
CNTLP    DS    ØH
         CLI   Ø(R1),C' '         End of byte count?
         BE    CNTDONE            Yes - wrap that up
         MVC   Ø(1,R14),Ø(R1)     Copy the next transfer count byte
         LA    R1,1(,R1)          Point to next source byte
         LA    R14,1(,R14)        Point to next target byte
         LA    R15,1(,R15)        Add one to byte count length
         B     CNTLP              Check for more
CNTDONE  DS    ØH
         ST    R15,CNTLEN         Save the length
         B     GETLOOP            Get next record
INDONE   DS    ØH
         OI    CLOSELST,X'8Ø'     Set parm bit on
         CLOSE (DCB1WRK),MODE=31,MF=(E,CLOSELST) Close the dataset
         TM    FLAG1,FTPOK        FTP success?
         BNO   RETURNØ8           No - we're done
         MVC   WTOGWRK(WTOGLN),WTOGLS Copy WTO model
         MVC   WTOGWRK+4(L'MSGØ1),MSGØ1 Copy first part of message
         MVC   WTOGWRK+4+L'MSGØ1(L'BYTECNT),BYTECNT Copy byte count
         WTO   MF=(E,WTOGWRK)     Issue the WTO
**       WTO   'FTPLINK - FTP completed successfully'
*-------------------------------------------------------------------*
*  If we get here, the FTP operation to the target system completed *
*   successfully.                                                   *
*                                                                   *
*  This is where you would insert logic for successful post-FTP     *
*   functions.                                                      *
*-------------------------------------------------------------------*
RETURN   DS    ØH
         L     R3,SAVEAREA+4      Copy incoming savearea address
         LR    R1,R13             Get current working storage addr
         STORAGE RELEASE,LENGTH=WORKLEN,ADDR=(R1)
         LR    R13,R3             Reload incoming savearea address
         LM    R14,R12,12(R13)    Restore registers
         XR    R15,R15            Set return code
         BR    R14                Return
*-------------------------------------------------------------------*
```

```
RETURN08 DS   0H
*------------------------------------------------------------*
*                                                            *
*  If a PARM='(EXIT' was specified on the EXEC statement and FTP  *
*  experienced a failure condition, the return code from FTP will *
*  be non-zero and it will contain specific information about the *
*  failure - namely the FTP subcommand that caused the failure and *
*  the corresponding reply code. Check for a non-zero return code *
*  and format an appropriate output message if this condition     *
*   exists.                                                   *
*                                                            *
*------------------------------------------------------------*
         L    R15,FTPRC          Get return code from FTP
         LTR  R15,R15            A non-zero return code?
         BZ   NOFTPRC            No - bypass message
         CVD  R15,DBL1           Convert to decimal
         L    R15,DBL1+4         Load significant part
         SRL  R15,4              Dump the 'sign'
         ST   R15,DBL2           Save the return code
         UNPK DBL1(9),DBL2(5)    Unpack the value
         NC   DBL1(8),=8X'0F'    Clear the high order nibbles
         TR   DBL1(8),=C'0123456789' Make the value readable
         MVC  WTOGWRK(WTOGLN),WTOGLS Copy WTO model
         MVC  WTOGWRK+4(L'MSG02A),MSG02A Copy first part of message
         IC   R1,DBL1+3          Get first byte of subcmd code
         N    R1,=X'0000000F'    Turn off first nibble
         MH   R1,=H'10'          Multiply by 10
         IC   R0,DBL1+4          Get second byte of subcmd code
         N    R0,=X'0000000F'    Turn off first nibble
         AR   R1,R0              Add to first byte
         LA   R15,SUBCMDTB       Get address of subcommand table
         LA   R14,CMD255         Get end of table address
SUBCMDLP DS   0H
         CR   R15,R14            Subcommand no good?
         BNL  SUBCMDNG           Yes - format message
         CH   R1,0(,R15)         A subcommand match?
         BE   SUBCMDOK           Yes - format message
         LA   R15,CMDENT(,R15)   Point to next entry
         B    SUBCMDLP           Check it out
SUBCMDNG DS   0H
         LA   R15,CMD99          Get catch-all subcmd address
SUBCMDOK DS   0H
         XR   R14,R14            Clear R14
         ICM  R14,B'0011',2(R15) Get length of subcommand
         BCTR R14,0              Reduce by one for EX
         EX   R14,SUBCMDMV       Copy subcommand
         LA   R15,WTOGWRK+4+L'MSG02A+1(R14) Point to new target area
         MVC  0(L'MSG02B,R15),MSG02B Copy in second part of message
         MVC  MSG02OF1(2,R15),DBL1+3 Copy the subcommand code
         MVC  MSG02OF2(3,R15),DBL1+5 Copy the FTPD reply code
         WTO   MF=(E,WTOGWRK)       Issue the WTO
NOFTPRC  DS    ØH
         L     R3,SAVEAREA+4        Copy incoming savearea address
         LR    R1,R13               Get current working storage addr
         STORAGE RELEASE,LENGTH=WORKLEN,ADDR=(R1)
         LR    R13,R3               Reload incoming savearea address
         LM    R14,R12,12(R13)      Restore registers
         LA    R15,8                Set return code
         BR    R14                  Return
*-----------------------------------------------------------------*
RETURN12 DS    ØH
         L     R3,SAVEAREA+4        Copy incoming savearea address
         LR    R1,R13               Get current working storage addr
         STORAGE RELEASE,LENGTH=WORKLEN,ADDR=(R1)
         LR    R13,R3               Reload incoming savearea address
         LM    R14,R12,12(R13)      Restore registers
         LA    R15,12               Set return code
         BR    R14                  Return
*-----------------------------------------------------------------*
LINKLS   LINK  SF=L
LINKLN   EQU   *-LINKLS
*-----------------------------------------------------------------*
OUTPUT   DCB   MACRF=(GM),DDNAME=OUTPUT,DSORG=PS,EODAD=INDONE
DCB1LN   EQU   *-OUTPUT
*-----------------------------------------------------------------*
WTOGLS   WTO   '                                                   x
                                                                   x
                    ',MF=L
WTOGLN   EQU   *-WTOGLS
*-----------------------------------------------------------------*
MSGØ1    DC    C'FTPLINK - FTP operation completed.  Byte transfer coux
               nt was '
*-----------------------------------------------------------------*
MSGØ2A   DC    C'FTPLINK - Non-zero FTP return code.  FTP subcommand '
MSGØ2B   DC    C' (command code xx).  FTPD reply code xxx.'
MSGØ2OF1 EQU   15                   Offset for command code
MSGØ2OF2 EQU   37                   Offset for FTPD reply code
*-----------------------------------------------------------------*
CMDENT   EQU   16                   Length of following table entries
SUBCMDTB DS    ØD
*        +CMD cd +CMD ln+SUBCOMMAND name+
*        ------- ------ --------------
CMDØØ    DC    AL2(ØØ),AL2(5),CL12'NOCMD'
CMDØ1    DC    AL2(Ø1),AL2(9),CL12'AMBIGUOUS'
CMDØ2    DC    AL2(Ø2),AL2(1),CL12'?'
CMDØ3    DC    AL2(Ø3),AL2(7),CL12'ACCOUNT'
CMDØ4    DC    AL2(Ø4),AL2(6),CL12'APPEND'
CMDØ5    DC    AL2(Ø5),AL2(5),CL12'ASCII'
CMDØ6    DC    AL2(Ø6),AL2(6),CL12'BINARY'
CMDØ7    DC    AL2(Ø7),AL2(2),CL12'CD'
```

```
CMDØ8  DC  AL2(Ø8),AL2(5),CL12'CLOSE'
CMDØ9  DC  AL2(Ø9),AL2(3),CL12'TSO'
CMD1Ø  DC  AL2(1Ø),AL2(4),CL12'OPEN'
CMD11  DC  AL2(11),AL2(5),CL12'DEBUG'
CMD12  DC  AL2(12),AL2(7),CL12'DELIMIT'
CMD13  DC  AL2(13),AL2(6),CL12'DELETE'
CMD14  DC  AL2(14),AL2(3),CL12'DIR'
CMD15  DC  AL2(15),AL2(6),CL12'EBCDIC'
CMD16  DC  AL2(16),AL2(3),CL12'GET'
CMD17  DC  AL2(17),AL2(4),CL12'HELP'
CMD18  DC  AL2(18),AL2(7),CL12'LOCSTAT'
CMD19  DC  AL2(19),AL2(4),CL12'USER'
CMD2Ø  DC  AL2(2Ø),AL2(2),CL12'LS'
CMD21  DC  AL2(21),AL2(7),CL12'MDELETE'
CMD22  DC  AL2(22),AL2(4),CL12'MGET'
CMD23  DC  AL2(23),AL2(4),CL12'MODE'
CMD24  DC  AL2(24),AL2(4),CL12'MPUT'
CMD25  DC  AL2(25),AL2(4),CL12'NOOP'
CMD26  DC  AL2(26),AL2(4),CL12'PASS'
CMD27  DC  AL2(27),AL2(3),CL12'PUT'
CMD28  DC  AL2(28),AL2(3),CL12'PWD'
CMD29  DC  AL2(29),AL2(4),CL12'QUIT'
CMD3Ø  DC  AL2(3Ø),AL2(5),CL12'QUOTE'
CMD31  DC  AL2(31),AL2(6),CL12'RENAME'
CMD32  DC  AL2(32),AL2(8),CL12'SENDPORT'
CMD33  DC  AL2(33),AL2(8),CL12'SENDSITE'
CMD34  DC  AL2(34),AL2(4),CL12'SITE'
CMD35  DC  AL2(35),AL2(6),CL12'STATUS'
CMD36  DC  AL2(36),AL2(9),CL12'STRUCTURE'
CMD37  DC  AL2(37),AL2(7),CL12'SUNIQUE'
CMD38  DC  AL2(38),AL2(6),CL12'SYSTEM'
CMD39  DC  AL2(39),AL2(5),CL12'TRACE'
CMD4Ø  DC  AL2(4Ø),AL2(4),CL12'TYPE'
CMD41  DC  AL2(41),AL2(3),CL12'LCD'
CMD42  DC  AL2(42),AL2(7),CL12'LOCSITE'
CMD43  DC  AL2(43),AL2(4),CL12'LPWD'
CMD44  DC  AL2(44),AL2(5),CL12'MKDIR'
CMD45  DC  AL2(45),AL2(6),CL12'LMKDIR'
CMD46  DC  AL2(46),AL2(8),CL12'EUCKANJI'
CMD47  DC  AL2(47),AL2(8),CL12'IBMKANJI'
CMD48  DC  AL2(48),AL2(7),CL12'JIS78KJ'
CMD49  DC  AL2(49),AL2(7),CL12'JIS83KJ'
CMD5Ø  DC  AL2(5Ø),AL2(9),CL12'SJISKANJI'
CMD51  DC  AL2(51),AL2(4),CL12'CDUP'
CMD52  DC  AL2(52),AL2(5),CL12'RMDIR'
CMD53  DC  AL2(53),AL2(7),CL12'HANGEUL'
CMD54  DC  AL2(54),AL2(7),CL12'KSC56Ø1'
CMD55  DC  AL2(55),AL2(8),CL12'TCHINESE'
CMD56  DC  AL2(56),AL2(7),CL12'RESTART'
CMD57  DC  AL2(57),AL2(4),CL12'BIG5'
```

```
CMD58  DC  AL2(58),AL2(5),CL12'BLOCK'
CMD59  DC  AL2(59),AL2(8),CL12'COMPRESS'
CMD6Ø  DC  AL2(6Ø),AL2(4),CL12'FILE'
CMD61  DC  AL2(61),AL2(5),CL12'PROXY'
CMD62  DC  AL2(62),AL2(6),CL12'RECORD'
CMD63  DC  AL2(63),AL2(8),CL12'SCHINESE'
CMD64  DC  AL2(64),AL2(5),CL12'STREAM'
CMD65  DC  AL2(65),AL2(4),CL12'GLOB'
CMD66  DC  AL2(66),AL2(6),CL12'PROMPT'
CMD67  DC  AL2(67),AL2(4),CL12'UCS2'
CMD99  DC  AL2(99),AL2(7),CL12'UNKNOWN'
CMD255 DC  AL2(255),AL2(6),CL12'TBLEND'
*-----------------------------------------------------------------*
       LTORG ,
*-----------------------------------------------------------------*
*  Executed instructions.                                         *
*-----------------------------------------------------------------*
SUBCMDMV MVC  WTOGWRK+4+L'MSGØ2A(*-*),4(R15) Copy subcommand
*-----------------------------------------------------------------*
*  Dynamic storage DSECT.                                         *
*-----------------------------------------------------------------*
WORKAREA DSECT
SAVEAREA DS   18F              Savearea
PARMADDR DS   F                Incoming parm address
FTPRC    DS   F                Return code from FTP
DBL1     DS   2D               Work area
DBL2     DS   2D               Work area
LINKWRK  DS   ØD,CL(LINKLN)    LINK SF=E work area
OPENLST  OPEN (,),MODE=31,MF=L   OPEN MF=E work area
CLOSELST CLOSE (,),MODE=31,MF=L  CLOSE MF=E work area
FLAGS    DS   ØF
FLAG1    DS   X                Flag byte
FTPOK    EQU  X'8Ø'            FTP success message flag
BYTECNT  DS   CL4Ø             Byte count value savearea
CNTLEN   DS   F                Length of byte count
INBUFF   DS   CL133            Input record
DCB1WRK  DS   ØD,CL(DCB1LN)    DCB work area
WTOGWRK  DS   ØD,CL(WTOGLN)    WTO MF=E work area
WORKLEN  EQU  *-WORKAREA       Dynamic storage length
*-----------------------------------------------------------------*
RØ       EQU  Ø
R1       EQU  1
R2       EQU  2
R3       EQU  3
R4       EQU  4
R5       EQU  5
R6       EQU  6
R7       EQU  7
R8       EQU  8
R9       EQU  9
```

```
R1Ø      EQU   1Ø
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         END
```

Rudy Douglas
(Canada)

Using the REXX Sockets interface to monitor a z/OS system from the Web

IBM supplies two sample REXX programs in the TCP/IP SEZAINST library, namely RSSERVER and RSCLIENT for IP Version 4. There are also samples for IP Version 6, but this article concentrates on the IP Version 4 ones. They are all documented in the *z/OS Communications Server: IP Application Programming Interface Guide* manual (SC31-8788).

The sample programs demonstrate the use of the REXX Sockets interface, and are intended to be used together. The server is started from a dedicated TSO userid (or batch job), binds a socket for a specified port number, and then waits for incoming requests. The client is run from another session to send a request to that port; the request is in the form of a number, which the server interprets and responds to by sending back that number of lines of text, generated internally.

I started by selecting an unused port and setting this up on a single LPAR in order to have two TSO sessions communicate with one another. I then tried with the server and the client on different LPARs (which is to say, with different IP addresses). Once again I was successful, after modifying the client to use the IP address of the server LPAR.

That seemed to be about as far as I could go with the samples, until it suddenly occurred to me that there was surely no reason why the server had to be called only by the sample client program, nor indeed why the client program even had to be a REXX program running on a z/OS mainframe. Why not, I thought, have the client be the Web browser running on my PC?

I immediately set about testing that possibility, and was soon gratified to discover that there was no problem: the RSSERVER sample program could respond to a request from a client which was a Web browser on a PC in just the same way as to RSCLIENT.

I had to deal with one minor issue, however; namely, the obvious one, that the PC is an ASCII device, whereas the mainframe is EBCDIC. This meant that the request that RSSERVER received had to be translated into EBCDIC in order to be understood, and the response had, likewise, to be translated into ASCII or else the Web browser just displayed what appeared to be garbage.

I now had a slightly modified RSSERVER that could receive a request from the Web and send lines of text. Once again, I felt that this was a neat result, but not necessarily very useful. When I put more thought into it, however, I realized two things:

- First, that ‘lines of text’ is not a limiting concept in the world of the Web, for HTML pages themselves are really mostly just lines of text.
- Second, that once a REXX program has control on a z/OS mainframe, it can extract just about any data it desires from system control blocks.

This meant that I already had the basic skeleton for writing a system which would allow me to monitor almost any aspect of the mainframe directly from my Web browser.

As an illustration of the power of this concept, the following modifications can be made to the supplied RSSERVER program, which enable it to call another REXX program, which in turn returns to the browser a formatted HTML display of information about the active jobs running on the system.

I have also added an option to tell the server to shut itself down cleanly when it receives a ‘q’ command from the client.

In RSSERVER, find line 77:

```
port = '1952'                /* The port used for the service   */
```

and change the port number to one appropriate for your site. Then find the comment on line 138 (but note that the same comment appears again on line 157):

```
/* Get nodeid, userid and number of lines to be sent            */
```

and the comment on line 176:

```
/* Unknown event (should not occur)                             */
```

Delete all the lines between these comments, and add these in their place:

```
  if keyword='READ' & ts¬=s then do
    parse value Socket('Recv',ts) with len data .
    request = substr(data,6,4)
    command = substr(data,6,1)
    if command = '71'x then do                       /* ascii 'q' */
      say 'RSSERVER: Request to quit'
      call addsock(ts)
      msg = '2Ø52535345525645522Ø69732Ø7175697474696E672Ø'x
     /* Ascii   R S S E R V E R   i s   q u i t t i n g          */
      call Socket 'Send',ts,msg
      call Socket 'Close',ts
      call delsock(ts)
      signal Terminate
    end
    else do
      call addsock(ts)
    end
  end
  if keyword='WRITE' then do
    if request = '6A6F6273'x then do              /* ascii 'jobs' */
      call JOBINFO ts
    end
    call Socket 'Close',ts
    call delsock(ts)
  end
end
```

Finally, find the comment:

```
/* Terminate and exit                                         */
```

and add this label immediately after it:

```
Terminate:
```

The call to program JOBINFO is executed when the following URL is accessed from a Web browser, either by directly typing it at the Addressfield, or as a link from another page:

```
<http://your.mf.ipaddr:nnnn/jobs>http://your.mf.ipaddr:nnnn/jobs
```

where *nnnn* is the port number specified on line 77 of RSSERVER as noted above.

Here is my sample program JOBINFO, which displays the CSTOR and cpu time used by all jobs currently active on the system. In order to demonstrate some additional flexibility, the program will highlight in red any cpu time figure over 1,000 seconds:

```
/*---------------------------- Rexx ------------------------------*/
/* JOBINFO -- Rexx/Sockets Web Server HTML page generator          */
/*-----------------------------------------------------------------*/
 numeric digits 21
 parse arg ts
 call JOBIHDR ts
 cvt  = storage(d2x(16),4)
 rce  = storage(d2x(c2d(cvt)+c2d(x2c(Ø49Ø))),4)
 asvt = storage(d2x(c2d(cvt)+c2d(x2c(Ø22C))),4)
 asvu = storage(d2x(c2d(asvt)+c2d(x2c(Ø2Ø4))),4)
 maxu = c2d(asvu)
 addr = d2x(c2d(asvt)+c2d(x2c(Ø21Ø)))
 asve = storage(addr,4)
 do i = 1 to maxu
   unus = bitor(substr(asve,1,1),'7F'x)
   if unus = 'FF'x then
     nop
   else do
     jbn = d2x(c2d(storage(d2x(c2d(asve)+c2d(x2c(ØØAC))),4)))
     if jbn = Ø then do
       jbn = d2x(c2d(storage(d2x(c2d(asve)+c2d(x2c(ØØBØ))),4)))
     end
     jobn = storage(jbn,8)
     if jobn = 'INIT    ' then
       nop
```

```
    else do
      ejst = storage(d2x(c2d(asve)+c2d(x2c(ØØ4Ø))),8)
      ejst = c2d(ejst)/(4Ø96*1ØØØØØØ)
      srbt = storage(d2x(c2d(asve)+c2d(x2c(ØØC8))),8)
      srbt = c2d(srbt)/(4Ø96*1ØØØØØØ)
      asn  = storage(d2x(c2d(asve)+c2d(x2c(ØØ24))),2)
      rax  = storage(d2x(c2d(asve)+c2d(x2c(Ø16C))),4)
      fmct = storage(d2x(c2d(rax)+c2d(x2c(ØØ2C))),4)
      cfr  = format(c2d(fmct),6)
      cmb  = format(cfr/256,4,1)
      cput = format(ejst+srbt,6,2)
      call htmlset
    end
  end
  addr = d2x(x2d(addr)+4)
  asve = storage(addr,4)
 end
/*  Optionally place a call to an HTML footer routine, JOBIFTR, here  */
 return
/*-------------------------------------------------------------------*/
/* Set up fields for HTML                                             */
/*-------------------------------------------------------------------*/
 htmlset:
 e1 = '<tr><td width="1Ø%" valign="middle">'
 e2 = '<b><font face="verdana" size=2>'
 e3 = '<p align="left">' || jobn || '</b></font></td>'
 call htmlgen
 e1 = '<td width="1Ø%" valign="middle">'
 e3 = '<p align="right">' || cfr || '</b></font></td>'
 call htmlgen
 e3 = '<p align="right">' || cmb || '</font></td>'
 call htmlgen
 if cput > 999.99 then
   color = '"#DDØØØØ"'
 else
   color = '"#33CC33"'
 e2 = '<b><font face="verdana" size=2 color=' || color || '>'
 e3 = '<p align="right">' || cput || '</font></td></tr>'
 call htmlgen
 return
/*-------------------------------------------------------------------*/
/* HTML generation                                                    */
/*-------------------------------------------------------------------*/
 htmlgen:
 em = e1 || e2 || e3
 address linkmvs "C2ASCII em"
 call Socket 'Send',ts,em
 return
```

JOBINFO calls program JOBIHDR to write the HTML header section of the page. This header is set up in a disk file that JOBIHDR reads, which allows the user to make the header information as simple or complicated as desired (including the use of special framing and formatting, logos and banners) without having to alter the REXX code at all:

```
/*---------------------------- Rexx -----------------------------*/
/* JOBIHDR  -- Rexx/Sockets Web Server HTML page header           */
/*---------------------------------------------------------------*/
 trace o
 parse arg ts
 done = 'n'; am = ''
 do while done = 'n'
   "execio 1 diskr jobihdr"
   if rc = Ø then
     do
     parse pull em
     call htmlgen
     end
   else
     done = 'y'
 end
 "execio Ø diskr jobihdr (finis"
 return
/*---------------------------------------------------------------*/
/* HTML generation                                                */
/*---------------------------------------------------------------*/
 htmlgen:
 address linkmvs "C2ASCII em"
 call Socket 'Send',ts,em
 return
```

If so desired, a similar routine can be added to JOBINFO where indicated, to insert additional footer HTML. This optional program, JOBIFTR, would be almost identical to JOBIHDR, except for reading from DDNAME jobiftr.

Below is a sample of some very simple header HTML that establishes a table framework and column headings for the detail lines that JOBINFO will supply, and is placed in a member in a PDS that is accessible via the JCL shown later:

```
 <html><head><title>Job information</title></head>
 <b><font face="Verdana" size="2">
 <p align="center"><center><table border cellspacing=1 width=6ØØ>
```

```
<tr><td width="10%" valign="middle"><b><font face="verdana" size=2>
<p align="left">Jobname</b></font></td>
<td width="10%" valign="middle"><b><font face="verdana" size=2>
<p align="right">CSTOR frames</b></font></td>
<td width="10%" valign="middle"><b><font face="verdana" size=2>
<p align="right">Megabytes</b></font></td>
<td width="10%" valign="middle"><b><font face="verdana" size=2>
<p align="right">CPU seconds</b></font></td></tr>
```

The final piece of the jigsaw is the EBCDIC to ASCII conversion, for which I use the excellent routine C2ASCII, which was described in *MVS Update* issue 176 (May 2001), in the article entitled ‘Utilities for FTP’ (*C2ASCII is reprinted immediately after this article*).

The modified RSSERVER program can be started with the following JCL:

```
//RSSERVER EXEC PGM=IRXJCL,PARM='RSSERVER'
//STEPLIB  DD DSN=YOUR.LOADLIB,DISP=SHR
//SYSEXEC  DD DSN=YOUR.SYSEXEC,DISP=SHR
//JOBIHDR  DD DSN=YOUR.HTML(JOBIHDR),DISP=SHR
//* JOBIFTR  DD DSN=YOUR.HTML(JOBIFTR),DISP=SHR
//SYSTSPRT DD SYSOUT=X
//SYSTSIN  DD DUMMY
```

C2ASCII is linked into the STEPLIB dataset, the REXX programs RSSERVER, JOBINFO, and JOBIHDR (and an optional JOBIFTR) are placed in the SYSEXEC dataset, and the HTML read by JOBIHDR is placed in the appropriate PDS member. In my example, a PDS specified FB,80,3200 worked perfectly, but HTML lines have no intrinsic length limitation so a greater record length may be desired. I have tested with a PDS specified VB,27994,27998 with success.

I run RSSERVER as a started task, and shut it down from my Web browser by entering

```
<http://your.mf.ipaddr:nnnn/q>http://your.mf.ipaddr:nnnn/q
```

Naturally, the ‘jobs’ in

```
<http://your.mf.ipaddr:nnnn/jobs>http://your.mf.ipaddr:nnnn/jobs
```

can be replaced by any other characters, and RSSERVER can simply be extended to call additional programs based on JOBINFO.

The following is an extract from a Web page I've created as a menu for some different options I use:

```
<UL>
<LI><A HREF="http://your.mf.ipaddr:nnnn/ipld">IPL data</A></LI>
<LI><A HREF="http://your.mf.ipaddr:nnnn/sysi">System info</A></LI>
<LI><A HREF="http://your.mf.ipaddr:nnnn/jobs">Job stats</A></LI>
<LI><A HREF="http://your.mf.ipaddr:nnnn/enqs">ENQs</A></LI>
<LI><A HREF="http://your.mf.ipaddr:nnnn/q">Quit</A>, shut RSSERVER</LI>
</UL>
```

Patrick Mullen
Consultant (Canada)

C2ASCII – ASCII to EBCDIC conversion macro

This conversion macro is referred to in the previous article, 'Using the REXX Sockets interface to monitor a z/OS system from the Web' (pp 12-18).

EBCDIC TO ASCII CONVERSION

```
/* REXX */
/* */
/* This edit macro is designed to convert EBCDIC data to ASCII */
/* */
ADDRESS ISREDIT
'MACRO'
'(start) = LINENUM .ZF'
'(endit) = LINENUM .ZL'
DO point=start UNTIL point>=endit
  '(line) = LINE' point
  address linkmvs "C2ASCII line"
  'LINE' point '= (line)'
   END
"LOCATE 1"
EXIT 1
```

ASSEMBLER ROUTINE C2ASCII

```
*********************************************************************
* C2ASCII: CONVERT DATA TO ASCII
*********************************************************************
C2ASCII AMODE 31
C2ASCII RMODE ANY
C2ASCII CSECT
       BAKR 14,Ø
        LR   12,15
      USING C2ASCII,12
        L    1,Ø(1)
       LH   5,Ø(1)       * GET THE LENGTH OF THE PARAMETER
        LA   4,2(1)
       XLATE (4),(5),TO=A
          PR
          END
```

ASCII TO EBCDIC CONVERSION

```
/* REXX */
/* */
/* This edit macro is designed to convert ASCII data to EBCDIC */
ADDRESS ISREDIT
'MACRO'
'(start) = LINENUM .ZF'
'(endit) = LINENUM .ZL'
DO point=start UNTIL point>=endit
  '(line) = LINE' point
  address linkmvs "C2EBCDIC line"
  'LINE' point '= (line)'
   END
"LOCATE 1"
EXIT 1
```

ASSEMBLER ROUTINE C2EBCDIC

```
*********************************************************************
* C2EBCDIC: CONVERT DATA TO EBCDIC
*********************************************************************
C2EBCDIC AMODE 31
C2EBCDIC RMODE ANY
C2EBCDIC CSECT
       BAKR 14,Ø
        LR   12,15
      USING C2EBCDIC,12
```

```
          L    1,Ø(1)
         LH   5,Ø(1)       * GET THE LENGTH OF THE PARAMETER
          LA   4,2(1)
         XLATE (4),(5),TO=E
           PR
           END
```

Grid computing – the next step for TCP/SNA

Grid computing is now one of IBM's strategic, very high-profile initiatives. The first sentence of its 12 December 2003 press release entitled 'IBM Expands Grid and Autonomic Computing Services' read: "IBM continues to bring the business benefits of grid computing to clients with the introduction of expanded Grid and Autonomic Computing services." What's significant here is IBM's clear assertion about the business benefits of grid computing, thus showing (yet again) its conviction that grid computing technology is applicable not only to scientific and academic computing, but to the commercial sector as well. This is why it will become the next important technology in the TCP/SNA application arena.

Grid computing isn't new for IBM. For the last two years, all of its mainframe-related announcements have referred to grid computing, albeit typically in conjunction with a discussion of autonomic (Project eLiza) computing, as in the case of the above quoted press release. Although grid computing and autonomic computing are very different, stand-alone initiatives, IBM's penchant for talking about the two together relates to the fact that grid computing exemplifies certain facets of both the self-optimizing and the self-configuring disciplines of autonomic computing. Grid computing is really another form of client/server-oriented distributed computing. It's a means of exploiting unused computer resources (ie processor cycles, memory, or storage) on a cooperative, but typically temporary, basis. As such, from an IBM perspective, grid computing impinges on both workload management (as in the zSeries' Intelligent Resource Director (IRD)) and the whole sphere of 'on-demand' computing.

SCAVENGING COMPUTER RESOURCES

Although grid computing is not in any way limited to specific computer systems or types, it's the abundance of PCs in today's world that provides the clear motivation for pursuing this technology. In any enterprise with upwards of 50 PCs, the chances are that at any given time less than 40% of the combined processing power, memory, and storage will be in use. And that statistic is for normal working hours: the processing and memory resources of most desktop PCs in such enterprises are totally unused outside that time. Contemporary enterprises therefore have a surfeit of valuable but unused computing power that could be gainfully used if there were a systematic means for harnessing this power. This is where grid computing comes in.

IBM cites its experiences with Bowne & Co of New York (www.bowne.com), a company that, among other things, specializes in offering a comprehensive array of transactional and compliance-related services to clients for creating, managing, translating, and distributing finance-oriented, mission-critical documents (eg mandatory stock market reports). It refers to this service as 'Financial Print'. Given the various immovable but periodic deadlines associated with most financial documents (eg month-end, quarter-end, year-end, tax filing, etc), the demands for computing, memory, and printing capabilities at Bowne fluctuate wildly depending on the time of year.

Although IBM's capacity on-demand (CoD) features on its zSeries, pSeries, and iSeries machines can address this problem, capacity on demand doesn't help customers harness the unused computing resources they already have. Grid computing, however, does, and this is the solution that Bowne adopted.

Another well-known example of grid computing in the commercial world is provided by Pratt & Whitney (www.pratt-whitney.com), maker of jet engines. Pratt & Whitney needs extensive CPU resources and computer memory to perform complex air-flow simulations and stress test analysis. Rather than resorting to super-computers for these tasks as it used to, Pratt & Whitney now uses a grid computing network made up of 5,000 desktop workstations, spread across three of its locations.

THE DISPATCH MODEL WITH SETI@HOME

The Search for Extraterrestrial Intelligence (SETI) at Home (setiathome.ssl.berkeley.edu), orchestrated by Berkeley University, is another well-known example of grid computing. SETI is trying to detect signs of extraterrestrial life by analysing radio frequencies emanating from space to determine whether they contain non-random patterns that might have been generated by intelligent beings. The amount of radio frequency data being analysed is enormous, and even Berkeley, with all its cutting-edge computing power, doesn't have enough capacity to do the necessary number-crunching in 'real-time'. It therefore came up with a grid computing solution that would make use of the unused computing resources of PCs around the world, irrespective of whether they were home-based or installed within an enterprise.

Berkeley devised an ingenious grid computing client that works as a screen-saver by graphically displaying via a colourful, 3-D spectrum chart the radio frequencies currently being analysed by a particular workstation. It created screen-saver clients for a variety of desktop platforms, including Windows, Mac, OS/2, BeOS, and OpenVMS. Once installed, each screen saver communicates with a central SETI server at Berkeley, over the Internet, to request small 'units of work' representing some amount of radio frequency data that needs to be analysed according to the SETI algorithms. The SETI server provides each screen-saver client with appropriate 'units of work' using grid computing's 'request model' of processing.

With this type of 'request model' operation, when a grid computing client is ready to process a new unit of work, it contacts a designated server and indicates its availability. The server then sends it a unit of work for processing. The client processes this unit of work when the necessary processing resources are available. In the SETI@Home example, the client needs to be able to do the necessary processing in the current unit of work in non-contiguous 'bites' of time, as and when the machine it's running on is not being used for other work. When the client finishes processing that unit of work, it contacts the server again and uploads the completed results back to the server.

In the ‘dispatch model’, a central server contacts various clients with new units of work that they’re expected to process when they have the necessary time (eg unused CPU cycles) or resources (eg unused main memory). When a client has processed the unit of work previously dispatched to it, it returns the results to the server using a pre-agreed protocol. The server synthesizes the results received from various clients, according to a larger algorithm, while continuing to dispatch new units of work to clients that have successfully completed their prior tasks.

TYPES OF GRID

There are currently three main types of computational grid, each best suited to a different type of application:

1 Scavenging grids

2 Computational grids

3 Data grids.

Scavenging grids, as their name suggests, set out to scavenge for unused computer resources that can be profitably exploited by other applications. They are primarily implemented with desktop workstations (eg PCs), although this grid methodology could also be used with servers. All of the examples cited above fall into this scavenging category of grid computing.

By contrast, grid computing involving high-performance servers (eg mainframes or Unix systems) is typically realized using computational grids, enabling the workload management capabilities of these servers to be harnessed to support a pre-determined grid computing application.

With computational grids, there are pre-determined policies as to how various resources on the machines involved will be ‘roped in’ to perform grid-related work. It’s also worth noting that IBM’s WebSphere Application Server Version 5 now supports grid computing via the ‘server allocation’ feature that was introduced in May 2003. Server allocation extends load-balancing across application servers to the point where all WebSphere Application Servers installed within an enterprise, irrespective of the exact platforms on which they’re running, appear as a single, unified application execution environment to the ‘guest’ applications that run on these servers.

While scavenging and computational grids deal with the sharing of CPU cycles and computer memory, data grids address how unused storage on systems can be shared by other applications.

GRID COMPUTING USING WEB SERVICES

Those of you who are familiar with XML Web services should see some similarity in the distributed computing paradigms being advocated by grid computing and Web services. Web services postulates client/server distributed mechanisms so that new applications can avail themselves of value-added functionality from external sources. Grid computing, on the other hand, advocates a client/server distributed mechanism, so that applications can avail themselves of additional processing resources available from external sources. Once you see this relationship, you can even start to identify scenarios where both methodologies could work together in a synergistic manner.

A new e-business application that relies on Web services for additional functionality could at the same time use grid computing to realize the horsepower it needs to complete all of its processing. So, rather than relying on additional processors or memory using a capacity-on-demand scheme, corporations could, provided that they had the appropriate grid-centric workload management infrastructure, think about exploiting grid computing using the inevitable unused processing power that’s always on tap across all organizations.

Thanks to the Open Grid Services Architecture (OGSA), Web services has now assumed a pivotal role in grid computing. OGSA is a key standard endorsed in June 2002 by the Global

Grid Forum – one of the more influential bodies promoting this technology. OGSA has since been implemented in the Globus Toolkit 3 (the Globus Alliance – www.globus.org – is an academia-oriented initiative sponsored by IBM, Microsoft, Cisco, and NASA).

OPEN GRID SERVICES ARCHITECTURE

OGSA is a standard for the overall structure and services that are to be provided in future grid environments. With OGSA, all grid resources, whether logical or physical, are represented as services – where a service is considered to be a network-enabled resource that delivers a predefined functional capability that can be invoked and used through the exchange of specific messages. OGSA will go a long way towards helping to commercialize grid computing, by defining a clear and consistent framework that addresses the overall structure and services that are to be provided in future grid environments.

OGSA is built upon Web services. In effect, it melds together grid computing with Web services –- or, to be more precise, Web services-related technologies such as XML, SOAP, Web Services Description Language (WSDL), WS-Security, and WS-Coordination. The fundamental building block of OGSA is a 'grid service', where a grid service is essentially an enhanced Web service.

Everything to do with OGSA therefore revolves around Web services. So much so that Globus now positions OGSA as representing the evolution of grid computing towards a grid system architecture based on Web services concepts and technologies. In essence, OGSA leverages Web services to slowly but surely move grid computing towards a highly service-oriented architecture (SOA) model.

Grid computing has also done a grand job of back-filling the holes that we still have in the Web services 'fabric'. The basic problem here is that there are no predefined backbone services with regard to Web services. In other words, there are no standard services *per se* for the life-cycle management of a Web service. Life-cycle management is crucial in the case of grid services – mainly because they tend to be much more transient (ie separate units of work rather than multiple calls to the same piece of software).

OGSA shows the Web services camp some of the things that they should be thinking about. What's likely to happen, however, is that there will be more and more collaboration between the camps, with each borrowing inspiration and ideas from the other.

SUMMARY

Grid computing is neither new, a novelty, nor just a passing fad. It sets out to address how unused computer resources can be harnessed and put to good use. This is of particular interest since so much desktop computing power goes unused, even though enterprises feel that they need more powerful systems. IBM sees grid computing as an extension to its on-demand computing initiative, which is why it often combines grid computing with autonomic computing.

The new OGSA standard brings the best of Web services methodology into the grid arena, in the form of grid services. Grid services therefore become another class of killer application for Web services, alongside portal and smart-phone applications. The introduction of Web services into the grid arena also highlights intriguing scenarios in which Web services and grid services can work in tandem, each addressing a very different need of an application.

The bottom line of all of this is that grid computing is going to be an important technology in the IBM world – particularly in the context of large-scale, mission-critical applications such as those associated with TCP/SNA. Which makes this an excellent time to start learning about it.

Anura Gurugé
Strategic Consultant (USA)

Using SOAP on a regular basis

Although SOAP is inextricably linked with XML Web services, the technology is by no means restricted to Web services. Rather, SOAP (which, as of Version 1.2, is now a *bona fide* standard endorsed by the World Wide Web Consortium (W3C)) is a strategic, XML-based messaging technology. It also facilitates remote procedure call invocation. It can be thought of as a kind of poor man's equivalent of IBM's WebSphere MQSeries, albeit one that's totally XML-centric.

SOAP is the latest in a long line of distributed computing initiatives – which, in this context, includes CORBA as well as Microsoft's COM/DCOM, although it's not meant to totally displace either of these powerful, object-oriented methodologies. SOAP sets out to be a simpler alternative without all the high-end bells and whistles (eg automatic garbage collection).

SOAP is a messaging scheme that works by exchanging XML documents. It is formally characterized as a lightweight communications protocol for exchanging XML-based information between applications, in a decentralized, distributed environment such as the Web. Think of SOAP, to begin with, as 'XML datagrams'.

XML is a mechanism for describing the meaning and context of data, and is contingent on both sides (ie creator and subsequent reader or consumer) being privy to some common 'intelligence' (eg XML schema or DTD). XML's initial focus, understandably, was all about being flexible and extensible, and having the necessary industry- and application-specific 'vocabularies' (or dialects) to facilitate b2b e-business. But the real scope of XML is limited to XML documents – that is, plain text files containing data annotated with XML notation. XML doesn't deal with how XML documents can be exchanged between interested parties; that's where SOAP comes in. SOAP (which is no longer an acronym – ie SOAP stands for SOAP) provides a generalized and extensible document exchange capability for XML, which is totally independent of Web services.

THE NEED FOR SOAP

In order to appreciate the rationale for SOAP in the context of XML without involving Web services in any way, let's take a look at the supply chain management (SCM) scenario of a supplier wishing to notify some of its customer base, programmatically, about new pricing data. With today's expertise in XML and the ready availability of the appropriate industry-specific 'vocabularies', it wouldn't be difficult to create the necessary 'price update' XML document. But the challenge now becomes that of conveying this document to all the intended recipients in an automated, program-to-program manner.

It's obviously not enough to send the XML document over HTTP as an e-mail attachment or via ftp to each of the recipients. This doesn't address the fundamental, program-to-program criteria; nor does it offer XML-oriented, application-level 'transaction' coordination, payload security, or guaranteed delivery.

SOAP sets out to address all of these distributed computing-related, message delivery requirements. SOAP can sustain any type of XML messaging application need, including those for one-way messaging, multicasting (or broadcasting), request-response interactions, and coordinated, sequential workflow progressions. It also includes the option of a simple but powerful RPC-type mechanism, allowing you to conduct XML-based, program-to-program, request/response-oriented RPC-like transactions that can be visualized as follows:

- placeOrder()
- getCreditRating()
- findCurrentWeather()
- getStockPrice()
- updatePurchaseOrder()
- obtainShipDate()

It is the powerful, platform- and programming-language-independent RPC mechanism of SOAP that's of special interest in the case of Web services. This RPC capability highlights the potential relationship between XML documents, SOAP, and Web services because each of the procedure calls shown above could be to a Web service that performs that function. Although SOAP is invariably associated with this type of RPC mode operation, it's important to note that this is only one of the messaging modes that can be used with SOAP. In reality, RPC mode representation is an optional SOAP capability.

THE RPC CAPABILITY

What SOAP really does when it comes to RPCs is provide a generalized mechanism for representing and encapsulating RPCs within SOAP messages. In theory, any of the schemes used by today's programming languages to define and invoke procedures or object-oriented 'methods' can be mapped and represented by a SOAP message, which would typically be delivered to the remote 'server' via an HTTP POST command.

Let's take as an example the 'findCurrentWeather' Web service, which, when given a zip code, returns an XML document that contains the current temperature, anticipated temperature range, current disposition (eg cloudy), and so on. A SOAP-based call to this service, which is hypothetically assumed to be available as a method at 'www.your-weather.com', would look as follows:

```
<w:findCurrentWeather xmlns:w="http://www.your-weather.com/">
  <w:sZipCode xsi:type="xsd:string">03249</w:sZipCode>
</w: findCurrentWeather>
```

The response to this call will be returned in another SOAP message.

A MORE COMPLEX SOAP EXAMPLE

Now let's look at the overall structure of a representative, but relatively simple, SOAP-based transaction. Imagine a hypothetical, on-line book ordering scenario between a book retailer and a publisher realized via an 'eOrder' SOAP request.

The SOAP request sent from the book retailer to the publisher will contain retailer identification, PO details, and information about the book. To achieve this, our customer will send the following ‘orderItem’ SOAP request to the supplier, which includes information such as the customer identification (ie RetailID), the item number, item name, item description, quantity ordered, wholesale price, and so on – in this case using HTTP as the transport mechanism:

```
POST /Orders HTTP/1.1
Host: www.megapublisher.com
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version=‘1.Ø’ ?>
<soap:Envelope xmlns:soap="http://www.w3.org/2ØØ3/Ø5/soap-envelope">
 <soap:Header>
  <o:eOrder
      xmlns:o="http://www.megapublisher.com/orders"
      soap:encodingStyle="http://megapublisher.com/encoding"
       soap:mustUnderstand="true">
  </o:eOrder>
 </soap:Header>
 <soap:Body>
  <m:onlineOrder>
   soap:encodingStyle="http://www.w3.org/2ØØ3/Ø5/soap-encoding"
      xmlns:m="http:// www.megapublisher.com/orders/">
  <m:eOrder xmlns:m="http:// www.megapublisher.com/orders">
   <m:retailerCode>UKØ216987</m:code>
   <m:retailerName>TechBooks</m:retailerName>
   <m:invoiceNumber>MgTec533</m:invoiceNumber>
   <m:isbnNumber>1-55558-28Ø-X</m:isbnNumber>
   <m:author>Guruge</m:author>
   <m:title>Corporate Portals</m:title>
   <m:quantity>1ØØ</m:quantity>
   <m:shipping>standard</m:shipping>
  </m:onlineOrder>
 </soap:Body>
</soap:Envelope>
```

The above SOAP request, delivered to the book publisher (ie ‘MegaPublisher.com’) via HTTP, will result in the ‘eOrder’ method being invoked at: www.megapublisher.com/orders/. SOAP itself doesn’t specify, or care, how this method is implemented. It also doesn’t get involved in how this request is processed. Its express goal is just to deliver the request, in the form of an invocation, to the intended remote method, along with the enclosed input data. The receiver, in this case the HTTP server at ‘MegaPublisher.com’, could use a standard CGI script, or invoke a Java servlet or .NET process, to process this order and return an appropriate acknowledgement (eg order confirmation number and an estimated delivery date).

SOAP MESSAGES AND NAMESPACES

Even a cursory examination of the above SOAP example will show that it consists of four distinct parts. The first four lines, starting with ‘POST’, talk about HTTP, identify the (remote) host, and specify a content type for the message. These four SOAP transport-related lines are known as the SOAP ‘binding’ (or, in this instance, the SOAP-HTTP binding, given that the transport is http). Within this, the first two lines, which are HTTP-specific, are known as the HTTP request.

Straight after the XML declaration, which comes next, you can find the start of a SOAP envelope. This envelope, which in essence subsumes the entire SOAP message, is in turn made up of two parts: the SOAP header and the SOAP body. All SOAP messages conform to this overall structure.

In addition, the abundance of colons in the above example (since each element name is prefixed, as well as the four occurrences of the key-word ‘xmlns’) indicates that XML namespaces are an integral concept within SOAP. XML namespaces is the standard XML mechanism to ensure that the meaning of XML elements with regard to a given XML document are not mistaken or misinterpreted. The prefixed element names with regard to a specific XML namespace ensure the uniqueness of XML elements within a particular context.

This explains the significance of XML namespaces with regard to SOAP. Since SOAP is a messaging scheme, the underlying implication is that an XML document transported using SOAP will be read and interpreted at a different site from where the original XML document was originally conceived. In addition, a recipient could receive XML documents from different sources that contain the same elements, but each with a slightly different nuance. The use of XML namespaces eliminates such ambiguity and makes sure that a recipient of an XML document has a way to unequivocally validate the intent of the elements contained in that document.

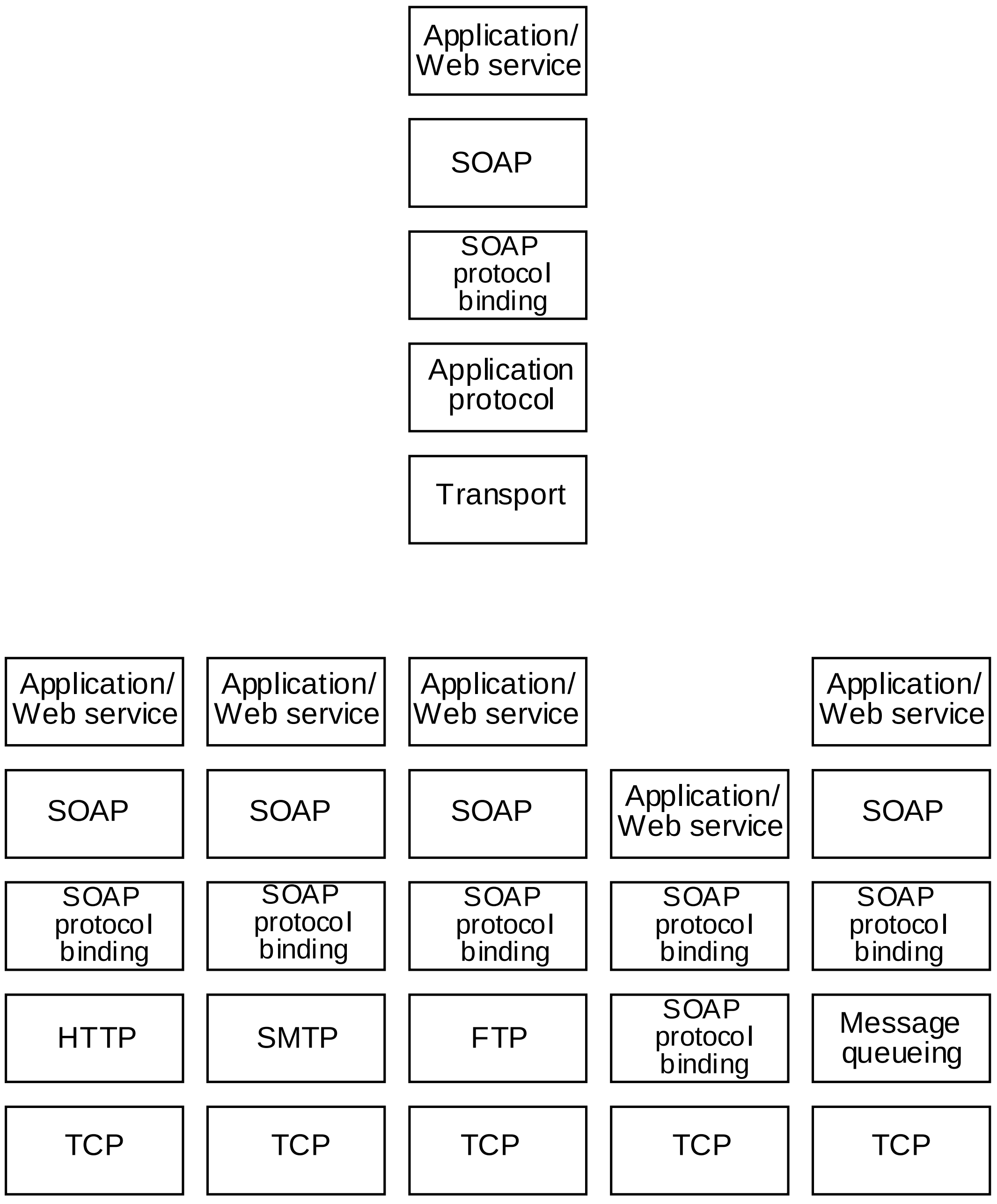

Figure 1: Layering of SOAP on top of the transport layer

THE KEY ATTRIBUTES OF SOAP

The primary attributes of SOAP can be summarized as follows:

- SOAP is an XML-based messaging scheme.
- SOAP works between applications.
- SOAP can readily be used between disparate platforms.
- SOAP supports the encapsulation and remote delivery of RPCs.
- SOAP is programming language agnostic.
- SOAP enables XML to be communicated over HTTP.
- SOAP, however, is not restricted to HTTP.
- SOAP relies heavily on XML namespaces.
- SOAP is what is used by Web services for their I/O operations.
- SOAP is also considered to be what remotely invokes Web services.

Given these attributes, it's easy to surmise the relationship between SOAP and Web services. SOAP, at least for the time being, is a fundamental, prerequisite building block for XML Web services. SOAP is the preferred mechanism for sending input to, and receiving output from, conventional XML Web services. It's therefore the underlying communications mechanism for today's Web services. Since a Web service requires input parameters in order to be activated, SOAP is also considered to be what invokes a Web service, given that it is what delivers the input parameters.

XML OVER HTTP

Contrary to popular belief, SOAP is not a transport mechanism but rather a messaging mechanism which is meant to be used on top of standard transport protocols. The layering of SOAP on top of the transport layer is clearly shown in Figure 1 (where

SOAP can be used by ‘standard’ applications as well as XML Web services). SOAP can be used across HTTP, TCP, SMTP, ftp, message queueing (eg IBM’s WebSphere MQ), or even other RPC mechanisms. However, HTTP is the preferred and most widely used transport scheme for SOAP, in the context of Web services as well as other scenarios.

HTML over HTTP is what the Web is all about. But with SOAP, you can have XML over HTTP. The SOAP specification acknowledges this ‘made in heaven’ relationship aspect of SOAP and HTTP. Although it is stated that SOAP is not limited for use with HTTP, the only transport ‘binding’ shown in the original SOAP specification (known as the 1.1 specification) was that for HTTP.

Given that SOAP is a messaging scheme, which moreover has structures known as ‘envelopes’ and ‘payloads’, it has become accepted to use a postal analogy to describe SOAP, although the analogy is somewhat limited. Nonetheless it would be remiss not to at least mention it in passing, given its widespread usage within the industry. SOAP, according to this analogy, is all about the item that is to be mailed. It describes how that item (ie the payload) is to be packaged in a modular manner. That’s where the SOAP envelope comes in.

But SOAP doesn’t dictate how this envelope containing the payload (ie the message) gets delivered to the intended recipient. Nowadays, with conventional paper-based mail, you have multiple delivery options, including normal mail, express mail service (eg FedEx), and courier. These are transport options – the message is the same, but the delivery characteristics are very different. The same is true with SOAP and the various transports over which it can be used.

FIREWALL TRAVERSAL

When used on top of HTTP, SOAP messages can typically cross corporate firewalls and evade standard packet-filtering policies, since HTTP, as the basis for interacting with Web servers, is given free access. Ironically, one of the motivations for developing SOAP was the fact that when used with HTTP it could indeed be a powerful remote procedure call mechanism that would not be blocked by a corporate firewall; CORBA- and the DCOM-based distributed computing approaches needed specific ports to be opened in the firewall in order for them to get through. Since network administrators are invariably reluctant to open up new ports on firewalls, this was yet another problem that beset the CORBA and DCOM approaches. Suffice to say that SOAP's ability to get through firewalls using HTTP is now a source of some concern, and one of the security-related issues that has to be dealt with in the context of Web services.

However, despite how it may seem at first sight, this firewall traversal capability is not an overt security exposure. This is because just getting one or more SOAP messages through a firewall is unlikely in itself to pose any kind of threat or cause any damage. SOAP always works in application-to-application, sender-to-receiver mode. This means that a SOAP message that crosses a firewall is simply a harmless 'lost transmission' unless it is received and acted upon by an application (or Web service) running behind the firewall. The secret for enforcing SOAP-level security is to ensure that any SOAP-capable application (or Web service) deployed behind a firewall is authorized, trusted, and regularly monitored.

SOAP PROVIDES INSPIRATION FOR WEB SERVICES

SOAP predates the advent of Web services. Indeed, it was really the creation of SOAP that got people thinking about the possibility of Web services. The origins of SOAP can be traced back to XML-RPC. XML-RPC was developed in early 1998 by a few visionaries working for Userland Software, DevelopMentor, and Microsoft (with the names of Dave Winer of Userland Software and Don Box of DevelopMentor inextricably associated with the development of XML-RPC and its influence on the creation of SOAP). XMP-RPC, as its name suggests, is a simple RPC mechanism that is realized using XML over HTTP. XML-RPC demonstrated that XML's scope didn't have to be limited to that of representing the exact meaning of a structured document. With XML-RPC, XML could also serve as the basis for a powerful, standards-based, distributed transaction processing scheme. XML-RPC is still in use today, although SOAP has in many ways supplanted it as its more strategic and widely known descendant.

Following XML-RPC, Microsoft and DevelopMentor went on to investigate transport-independent, XML-based messaging schemes that could be used for distributed computing. They were striving for a mechanism that was simpler to deploy and use than either DCOM or CORBA, which were considered at that time to be the long-term solutions for component-based, distributed computing, with the former being Microsoft-specific while the latter was 'open' and vendor-independent. The resulting specification, which was known as SOAP 1.0, was ready in September 1999. In order to garner sufficient market backing for this specification, Microsoft sought other partners. The result was the SOAP 1.1 specification, which was authored by representatives from Microsoft, DevelopMentor, IBM/Lotus, and UserLand Software.

The 1.1 specification was submitted to W3C on 8 May 2000. Collaborating on SOAP provided IBM and Microsoft with the inspiration and impetus to flesh out the notion of XML Web services that would use this new XML-based messaging scheme as their I/O mechanism. IBM, Microsoft, and Ariba thus went on to create the specifications for WSDL and UDDI, which were made public six months after the unveiling of SOAP.

In addition to the four companies that 'authored' it (keeping in mind that Lotus is a division of IBM), the submission of the SOAP 1.1 specification to W3C was further endorsed by other then big names in the industry, including Ariba, Compaq, H-P, SAP, IONA Technologies, and CommerceOne. Given this wide and influential industry backing, W3C did not subject this specification to the rigorous ratification process that is typically the norm. Bowing to the momentum that SOAP had already picked up by that stage, and the fact that SOAP implementations were already in progress, particularly from Microsoft (ie

Microsoft's SOAP Toolkit 1.0, which was available in the Summer of 2000) and the Java camp (led by IBM and Apache), W3C accepted SOAP 1.1 as a *de facto* industry standard.

SOAP 1.1 was not as precise as some would have wished, and there were some interoperability issues between different implementations of the 1.1 specification. This led to the W3C's XML Protocol Working Group initiating work on a SOAP 1.2 specification that attempted to address the nearly 400 technical and editorial issues cited against the original specification.

SOAP IMPLEMENTATIONS

In order to be able to use SOAP in practice, in the real world, irrespective of whether it is for use with Web services or with non-Web services-related software, you need an actual software implementation of the SOAP specification. However, this is not going to be an impediment, given that there are currently over 70 SOAP implementations covering all popular hardware and software permutations. A relatively comprehensive list of current SOAP implementations can be found at: www.xmethods.net.

In general, developers will not set out to directly generate and receive SOAP messages. Instead, when developing Web services that will use SOAP for their I/O operations, they would typically opt to use a SOAP Toolkit, an application development 'studio' product (eg IBM's WebSphere Studio Application Developer, BEA WebLogic Workshop, and Microsoft's Visual Studio .NET 2003), or a Web services-specific toolkit (eg IBM's Web Services Toolkit (WSTK) or Microsoft's WSTK) to help them create, parse, and manage the necessary SOAP messages.

SOAP-based Web services (or other applications) once developed using a toolkit or a 'studio' will typically be deployed, at least on the server-side, on top of application servers (eg IBM's WebSphere Application Server, BEA's WebLogic Server, Microsoft's .NET Framework, Apache Tomcat, Sun ONE, and the 'open-source' Jboss) that in turn will support SOAP-based messaging over a variety of different transports. On the client side, SOAP support can generally be obtained in two distinct ways depending on the platform in question. Microsoft's .NET Framework, which is available for Windows XP, Windows CE, .NET 4.2, and Windows Mobile 2003, includes built-in support for SOAP. Microsoft's technology overview for .NET Framework 1.1 starts off with a statement that runs as follows: "the .NET Framework 1.1, used for building and running all kinds of software, including Web-based applications, smart client applications, and XML Web services-components facilitate integration by sharing data and functionality over a network through standard, platform-independent protocols such as XML (Extensible Markup Language), SOAP, and http". If, instead, Java- or C++-based support is required on a client, you would typically include the necessary SOAP libraries, provided by the original development tool, as an overall part of the Java or C++ code that would be downloaded to the client.

Some of the best known and most widely used SOAP implementations include the following:

- Microsoft's SOAP Toolkit 2.0
- IBM's Java-based SOAP4J, which went on to become the Apache SOAP project
- SOAP::Lite for Perl
- EasySOAP++
- GLUE.

MICROSOFT'S SOAP TOOLKIT

Given its pioneering work on SOAP, it's not surprising that Microsoft was one of the first to offer a SOAP implementation. This was the so-called SOAP Toolkit 1.0, which was available in the summer of 2000, shortly after SOAP 1.1 had been formulated and submitted to the W3C. This initial Toolkit did not support WSDL (which was at that stage still being formulated), and only accommodated RPC-mode transactions over HTTP.

Version 2 of this toolkit, which received widespread publicity within the software development community and, as such, became a *de facto* standard for this genre, was officially available as of April 2001. Although Microsoft came out with a Version 3 in July 2002, it’s still the 2.0 toolkit that most developers have in mind when they think about SOAP-specific development options from Microsoft.

The SOAP Toolkit permits developers to add XML Web Service functionality to existing Microsoft COM-oriented applications and components. The Toolkit is heavily WSDL-centric. It includes a WSDL generator that will automatically generate WSDL descriptions of existing COM libraries. It can be used by itself or in conjunction with Microsoft Visual Studio .NET. The primary features of the 2.0 Toolkit include:

- A client-side component that permits applications to invoke SOAP-based Web service operations as described by the WSDL definition for that service.
- A server-side component that generates SOAP messages from COM object method calls according to the WSDL description, albeit only when augmented with Microsoft-specific Web Services Meta Language (WSML) files.
- The necessary code to transmit, read, ‘serialize’, and process SOAP messages.

The 3.0 Toolkit included support for sending and receiving attachments (eg pictures). Remember that this type of attachment handling essentially falls into the realm of SOAP bindings (eg SMTP with MIME), SOAP extensibility, and SOAP encoding. It’s therefore a legitimate implementation-related option. Microsoft sets out to further extend this capability by adding support for Direct Internet Message Encapsulation (DIME)-based attachments, where DIME is a new proposed standard, for use with SOAP, which (similar to MIME) will allow any file, of arbitrary type, to be attached to a SOAP message. There is also a generic type mapper to facilitate the mapping of complex data types to the necessary WSDL and WSML descriptions.

SOAP IN JAVA

On the Java side, Apache Axis, from The Apache Software Foundation (www.apache.org), the doyen of Web-oriented open-software, is now the 'gold standard' when it comes to SOAP implementations. Apache Axis, which includes support for WSDL as well as XML-RPC, supersedes the well-known Apache SOAP project. This project, in turn, got its impetus when IBM donated its SOAP4J implementation to Apache. Axis; this is positioned as a SOAP engine, is 1.2 compliant, and can effectively be thought of as Apache SOAP 3.0.

Axis not only offers new features – such as WSDL and XML-RPC – but also represents a total rewrite of the original SOAP4J code, as well as a move, within this code, from the display-oriented XML DOM API to the more appropriate, inter-program-oriented SAX API. The Axis implementation is also credited with being much faster than the Apache SOAP version.

The Axis SOAP engine is available as a simple, small-footprint, stand-alone server. It's also available as a 'plug-in' that can be seamlessly integrated into application (or servlet) servers. The Axis SOAP implementation is already in use by IBM, Apple, Borland, Macromedia, and Jboss, amongst others. IBM's market-leading WebSphere Application Server 5.0 has the Axis software built in. IBM's WSTK also uses the Axis software. Axis, like the Microsoft SOAP Toolkit, provides support for attachments but, whereas the Microsoft Toolkit is COM-centric, Axis is Java Bean oriented. Thus, for example, it will automatically 'serialize' Java Beans, with the added option of customizable mapping to specific XML fields or attributes.

SOAP INTEROPERABILITY

Interoperability, as mentioned earlier, was an issue with SOAP 1.1 implementations. Various efforts are therefore afoot to ensure that this will not continue to be a stumbling block down the road, particularly with 1.2-based implementations. The SOAP 1.2 specification therefore includes an 'Assertions and Test Collection' test suite. There's also the Web Services

Interoperability (WS-I) organization, founded by Accenture, BEA, H-P, IBM, Intel, Microsoft, Oracle, SAP, and Fujitsu in February 2002. Since then, Sun, Cisco, and many others have joined WS-I in various capacities.

WS-I is an open industry organization whose goal is to promote Web services interoperability across platforms, operating systems, and programming languages. WS-I sets out to work across the industry and standards organizations as an end-user-focused 'advocacy' that provides guidance about best practices regarding Web services and the standards on which they were based. One of its explicit charters is that of creating, promoting, and supporting generic protocols for the interoperable exchange of messages between Web services. Individual implementers such as Apache are also doing their part to foster interoperability. The Axis SOAP engine, for example, has been subjected to a Java-related interoperability test specified by Java creator and mentor Sun.

The bottom line here is that Web services development, deployment, or exploitation is in no way going to be hindered by the paucity of SOAP implementations or by interoperability fears. The fact that the SOAP specification doesn't currently address the needs of transaction processing scenarios could be an issue as more and more organizations opt to start exploiting Web services for production applications. For the time being, however, these issues can be effectively addressed by using SOAP over a transport scheme that does offer the necessary transactional processing oriented features (eg a cross-platform, message queuing scheme such as IBM's WebSphere MQ).

Most major toolkits and 'studios' targeted at Web service developers offer SOAP functionality for the application servers that will be used as the execution environments for Web services. This means that the choice as to which SOAP implementation an organization uses will invariably be dictated by the overall software development preferences and dictates of that organization. If it favours .NET-based software development, it will most likely start by looking at Microsoft's

Visual Studio .NET 2003, SOAP Toolkit 3.0, and WSDK options. If, on the other hand, it happens to be a Java-shop, it will most likely start with one of the popular Java studios and application servers, such as IBM's WebSphere or BEA WebLogic. Then there are all the other language-specific options for C++, Perl, and so on.

THE SOAP 1.2 INITIATIVE

SOAP 1.2 was expected, at a minimum, to do the following:

- Develop an 'envelope-oriented' encapsulation scheme for XML data so that it could be transferred in an interoperable manner between disparate systems, allowing for future extensibility and evolution in terms of distributed systems – particularly in terms of possible intermediary nodes that may occur between the transmitter and the receiver, where these intermediaries could be in the form of gateways, caches, or application-level 'proxies'.
- Ensure, with the cooperation of the IETF, an operating system- (and programming language-) independent means for representing the contents of the SOAP envelope when SOAP is used for RCP-related operations.
- Define a mechanism based on XML schema data types (eg xsd:string, xsd:integer, xsd:decimal and xsd:Boolean) to represent necessary data (where such a process for representing data so that it can be correctly interpreted at the remote end is referred to as 'data serialization').
- To define, yet again with the cooperation of the IETF, a non-exclusive transport mechanism that could be layered on top of HTTP.

The SOAP 1.2 specification, sanctioned as an official W3C 'recommendation', was made available on 24 June 2003. This W3C recommendation status makes SOAP 1.2 a *bona fide* standard. The 1.2 specification consists of two parts:

- SOAP Version 1.2 Part 1: Messaging Framework
- SOAP Version 1.2 Part 2: Adjuncts.

These two parts, which constitute the technical body of the specification, were edited by representatives from Canon, IBM, Microsoft, and Sun. This technical portion of the specification is augmented by an introductory primer, which is referred to as 'SOAP Version 1.2 Part 0: Primer'. This document was edited by a representative of Ericsson. The complete 1.2 document set, however, is considered to consist of one other document, the SOAP Version 1.2 Specification Assertions and Test Collection. This document, which is essentially a test plan for 1.2, was created by representatives from Active Data Exchange, AT&T, IONA Technologies, Oracle, Unisys, and W3C.

The Assertions and Test Collection is designed to foster interoperability between diverse 1.2 implementations. Given that interoperability was an issue with 1.1, it's easy to appreciate the motivation for this specification validating suite of tests. This document captures assertions found in the SOAP Version 1.2 Part 1 and Part 2 specifications, and provides a set of tests that indicate whether the assertions are properly implemented in a given SOAP implementation.

These tests are meant to help SOAP implementers check how well their creations comply with the actual specification. A SOAP 1.2 implementation that passes all of the tests specified in this document may claim to conform to the 24 June 2002 SOAP 1.2 Test Suite – this being the date that this document was accepted as a W3C recommendation. In theory, all those implementations that successfully pass the entire suite of tests contained in this document should be able to cleanly interoperate with each other without encountering unexpected exceptions.

However, the successful completion of this test suite doesn't necessarily guarantee total SOAP 1.2 compliance. The test suite admits up-front that it doesn't test all aspects, and in particular it doesn't test those facets of an implementation that are considered to reflect the core, mandatory SOAP 1.2

requirements spelt out in the specification. Despite this, however, it's definitely worthwhile having a test suite that sets out to validate a relatively large part of the overall specification.

XML INFOSET

On a related subject, the 1.2 specification also strives to be as unambiguous as possible, especially when describing the structure of the XML-based documents, yet again in order to minimize potential implementational variations that could result in interoperability issues. Part 1 of the specification that deals with the composition of SOAP messages therefore resorts to the use of 'XML Information Set' (XML InfoSet) conventions. XML InfoSet is a relatively new W3C standard that was formally ratified in October 2001.

XML InfoSet is designed to provide a consistent set of definitions for use in other specifications that need to refer to the information in a well-formed XML document. The XML InfoSet is an abstract dataset for XML documents. In effect, it's a guide for writing more rigorous XML, and emphasizes the use of XML namespaces to eliminate ambiguity. Given this mission to promote better structured XML documents, it's easy to understand its appeal to those crafting the SOAP 1.2 specification. With the use of InfoSet, SOAP 1.2 also shifts the data serialization (ie 'remote' data representation) issue to correspond with the transport that's to be used. It's therefore left to the specification for a transport binding to dictate the serialization scheme that will be used with that transport.

Other than the use of InfoSet, with its emphasis on namespaces, most of the other changes between 1.1 and 1.2 tend to be in the realm of technical refinement, and are somewhat esoteric. For example, with 1.1 it was possible to have other elements, known as 'trailers', after the payload-carrying body element of a SOAP message. In other words, there could be elements between </s:Body> and </s:Envelope>. SOAP 1.2 does not permit such trailers.

In order to be as flexible and extensible as possible, SOAP advocates a message transfer model where there can be a chain of SOAP-cognisant nodes between the originator and receiver of a message. Each node in the chain can process a part of the overall SOAP message. Nodes that perform some level of processing on a SOAP message are known as SOAP 'endpoints'. The processing performed by intermediary endpoints typically relates to header items within the SOAP message.

BOTTOM LINE

Despite its close association with XML Web services, SOAP is a strategic, stand-alone messaging mechanism with a powerful and useful RPC capability. It augments and complements XML by providing a standard means by which XML documents can be conveyed to one or more recipients in a systematic manner. Given that it can be used across many different transports, including HTTP, SOAP provides a standard means by which to transfer XML over HTTP. SOAP implementations are widely available for all popular computing platforms and, with Version 1.2, interoperability between disparate implementations should no longer be an issue. Given that SOAP, now as a *bona fide* W3C standard, is gaining increasing attention as an XML-centric, application-to-application messaging and RPC-invocation scheme, independent of its role with Web services, this is a good time to start brushing up on SOAP.

Anura Gurugé
Strategic Consultant (USA)

Automatic ftp to another AIX site

The utility presented here enables files to be transported from an AIX system to another server, without any ftp coding in a user application.

REQUIREMENTS

The following resources are required on each machine:

- One ftp user: ftp_user
- One filesystem: /ftp
- One ini file: /home/data/ftp.ini
- Two crontab entries: one for ftp to receive and one for ftp to send.

The following resources are generated by the application:

- One or more ftp-commandfile(s): triggerfile to activate the send or receive action.
- File(s) to be transported.

Also required is a user to send files to the other server. Although you can use root or another user, it's much better to use a dedicated user for ftp. This user will always be present, and it's the only one that will ftp files to another site.

The user on the receiving site is root: only this user can switch between several other users without giving passwords. It may need to do this in order to:

- Start up a database (user: oracle)
- Start applications owned by and using a specific owner, eg concurrent processes user = applmgr, logistic processes : user = logic1, development processes user = develop1

The structure of the filesystem /ftp is as follows:

```
                  |-- backup
       |-- receive -|
        |         |-- error
  /ftp  -|
        |         |-- backup
       |-- send   -|
                  |-- error
```

The ini file contains parameters for ftp: replace or append files, and what command has to be started if a command file appears, etc.

The crontab entries will start the ftp scripts – ftp_receive and ftp_send – every ten minutes.

The command files trigger a required send or receive action. The command file will be sent once all the files are present; only then can you be sure that all the files are complete.

HOW IT WORKS

File ftp.ini

Before you can send files, you have to make an entry in ftp.ini. This ini-file will be used by the ftp_receive and ftp_send scripts.

Two examples of ftp.ini are shown below.

On the send site:

```
        [test.cmd]
action=
cmd_bef_send=
cmd_aft_send=
description=load files into the remote database
name=test_load.cmd
path=
script=
```

On the receive site:

```
[test_load.cmd]
action=replace
cmd_bef_send=
cmd_aft_send=
description=load files from the master database
name=
```

Entry	*Possible values and explanation*	*Used by*
action=	replace: rewrite existing files append: append to existing files.	Receiver
cmd_bef_send=	ftp coding that will be involved before the ftp of the files is started.	Sender
cmd_aft_send=	ftp coding that will be included after the ftp of the data files.	Sender
name=	name of command file at the other site. It's possible to send the command file with another name, because at the other side the name of the original command file may be used for something else.	Receiver and sender
path=	path where ftp_receive has to put the files.	Receiver
script-=	command that will be executed after a succesful ftp at receive site.	Receiver

Figure 1: Entry values

```
path=/interface
script=test_load07
```

Figure 1 shows a list of entries and their values.

In order to use this utility some recoding is necessary. To simplify this, the ftp utility can execute scripts before and after an ftp. So you can split your coding and code only the creation of a command file.

Some applications will execute remote commands after an ftp. To make that possible, the entry ‘script’ has been added. This script seeks its datasets in a particular directory. To minimize rewriting the application, the entry ‘path’ has been added: move the files to that path.

EXAMPLE OF SENDING FILES

This example uses the ini files coded above.

The application has to put the files, which have to be transferred, in the /ftp/send directory. After that, it generates the command file test.cmd and puts this in the /ftp/send directory too.

At this point the application can stop.

The script ftp_send will be started from the crontab within ten minutes. The files will be sent to their destination in the directory /ftp/receive, and the command file will be transferred with the new name test_load.cmd. After a successful ftp, the script will move the transferred files into a back-up directory.

At the destination site, ftp_receive finds the command file test_load.cmd. The script searches in the ftp.ini and copies the files in /interface – existing files will be overwritten. The files are moved into /ftp/receive/backup. It then executes the script test_load with the parameter 07.

Command file

The command file is filled by the script that generates the files, and contains the destination and the filenames that have to be sent to the other site.

Example command file test.cmd

```
target_site
dump_file1.dmp
dump_file1.log
```

In this example, the files dump_file1.dmp and dump_files1.log *and* the command file test.cmd will be transferred to the site target_site.

The complete result after a successful ftp will be as follows:

At the sending site:

```
 /ftp/send/backup/dump_file1.dmp.date-time
 /ftp/send/backup/dump_file1.log.date-time
 /ftp/send/backup/test.cmd.date-time.ftp
 /ftp/send/backup/test.cmd.date-time
```

At the receiving site after ftp_receive:

```
 /interface/dump_file1.dmp
 /interface/dump_file1.log
 /ftp/receive/backup/dump_file1.dmp.date-time
```

```
/ftp/receive/backup/dump_file1.log.date-time
/ftp/receive/backup/test_load.cmd.date-time
```

Note that date-time is a date time stamp of the format: MMDD-hhmmss (MonthDay-HourMinuteSecond).

THE FILESYSTEM /FTP

The following code will create a filesystem /ftp of 100 Mb. If you want to make it larger, change 13 into the number you need.

To make a logical volume labelled ftp with a size of 13 PPs (ppsize of the vg volumegroup1 = 8 Mb):

```
mklv -y'ftp' -t'jfs' -x'25' volumegroup1 13
```

Create a normal journalled filesystem called ftp with a mount point /ftp in the previous logical volume:

```
crfs -v jfs -d'ftp' -m'/ftp' -A''`locale yesstr | awk -F: '{print $1}'`'' -p'rw' -
t''`locale nostr | awk -F: '{print $1}'`'' -a frag='4096' -a nbpi='4096' -a ag='8'
```

Mount the filesystem mount /ftp.

FTP_USER

The user will use the group ftp. If you want to use another group, eg system, use that one; if you want to use the group ftp and it doesn’t exist, create the group with the following command:

```
mkgroup -'A' ftp
```

The user is created by issuing the following command:

```
mkuser -a "id=220" "pgrp=ftp" "su=false" "home=/ftp" "gecos=ftp batch user" "fsize=-1"
"ftp_user"
```

Reset the password of ftp_user to a known one: nothing.

Press on the <ENTER> key in response to the next two questions asked by the command passwd.

PASSWD FTP_USER

Login as ftp_user and set the ftp_user password by typing the following command:

```
passwd
```

Create the adjusting directory structure in /ftp.

As root:

Change the owner of the ftp filesystem to ftp_user:

```
chown ftp_user:ftp /ftp
```

Become ftp_user and set up all the directories and files:

```
su - ftp_user
```

Fill the password file, replace “the ftp_user password” with the actual password of ftp_user

```
echo "the ftp_user password" >ftp_user_password
```

Create the directories:

```
mkdir receive
mkdir send
cd receive
mkdir backup
mkdir error
cd ../send
mkdir backup
mkdir error
```

Make it possible for everyone to put files in /ftp/send:

```
cd /ftp
chmod 777 send
```

The ftp of files can be right or wrong

- If it’s right:
  - At the sender, the files will be moved to /ftp/send/backup followed by a timestamp.
  - At the receiver. the files will be moved to /ftp/receive/backup followed by a timestamp.
- If it’s wrong:
  - At the sender, the problem may be that some files do

not exist, the ftp went wrong, script(s) do not exist, or an entry in ftp.ini is not there. The files will be moved to /ftp/send/error followed by a timestamp, and an administrator will need to resolve the error or wait until the site is up again, and resubmit the files to the ftp-system by using ftp_restart.

– At the receiver, the problem may be that the script which has to be executed does not exist, the entry in ftp.ini is not there, or the target directory is missing. The files will be moved to /ftp/receive/error followed by a timestamp. Once again, an administrator will need to resolve the problem.

The back-up directories are meaningful, and are created to ensure that the ftp can always be restarted.

Users are sometimes unsure exactly what data was loaded in their system. They can easily look into the flat files to find any data they're missing.

CRONTAB ENTRIES

We have to use two crontabs: one of the user ftp_user and one of the user root.

The crontab of the user ftp_user has just one entry:

```
Ø,1Ø,2Ø,3Ø,4Ø,5Ø * * * * /home/oper/ftp_send
```

The crontab of the user root has two entries added:

```
5,15,25,35,45,55 * * * * /home/oper/ftp_receive
Ø 4 * * 6 /home/oper/ftp_cleanup
```

SCRIPTS

The ftp_receive script has to be executed by the user root because the script, which has to be executed after the receive, may change from the user, eg Oracle. Only the user root can freewheel between users without knowing their passwords.

The ftp_send and ftp_receive scripts will be executed every ten minutes to trigger command files. The ftp_cleanup script will be executed every Saturday at 4 o'clock.

ftp_cleanup

The ftp_cleanup script will remove old files which exist in the /ftp/send/backup, /ftp/send/error, /ftp/receive/error, and /ftp/receive/backup directories. The script will also replace a triggerfile and log all the removed files.

ftp_send

The ftp_send script will read one or more command file(s). The first line is always the destination (eg target_site). The next lines are the files which have to be transferred, and each file will be checked to see whether it exists. If a file does not exist, the ftp will not take place and all the other files called in the command file will be moved into the error directory. If all the files are present, the ftp session will start and the files will be transferred. After the ftp, the script checks that all files have been sent to their destination. If everything is OK, the files will be moved to the back-up directory; otherwise, the files will be transported to the error directory and a message will be generated.

If ftp_send is already executing, the script ends. Once it has ended three times, it will generate a message; after ending four times without any action, it will give a serious warning.

ftp_receive

ftp_receive will check to ensure that all the files are present.

If any file is missing or the destination directory does not exist, or if the script which has to be executed does not exist, the files will be moved to the error directory and the script called in ftp.ini will not be started in the background.

Only if all files are present and everything else is ok will the files be copied to their target directory and moved to the back-up directory. After moving the datasets, the script called in ftp.ini is started in the background, and ftp_receive will end.

ftp_restart

The ftp_restart script can be used when an ftp session fails, for example when the destination cannot be reached.

You can start it by typing ftp_restart on the prompt. The script will return a screen containing the sessions and the time when they failed. You can choose one entry to restart. The files will be transferred to /ftp/send, and the ftp system will process the files in the next loop. Any error will be shown.

You can also add the script in the crontab. When started, this will restart all the failed sessions except those which cannot be restarted – for example where files are missing or the destination cannot be reached. After a session has been restarted a certain number of times, an error message will be produced and the ftp-session will be skipped.

An example crontab entry for ftp_user is shown below:

```
3,13,23,33,43,53 * * * * /home/oper/ftp_restart
```

THE CODING

All the scripts use the ‘script logging system’, which contains logmsg, logfile, and d_msg. These scripts are used to log all the processes.

(The coding scripts are available for download from our Web site. See the end of the article for further details.)

EXAMPLE LOGGING ENTRIES

ftp_receive

Example 1 : ftp_receive did find all files and started a command

```
15/Ø9/99 Ø3:Ø5:Ø1 24876 root   ftp_receive - ftp - bls_ecodet_goe_R1.cmd: chown
applmgr:dba /oraapplx/applmgr/product/1Ø7/sch/1.Ø/gldata/ijnp*.* started
15/Ø9/99 Ø3:Ø5:Ø2 24876 root   ftp_receive - ftp - bls_ecodet_goe_txtR1.cmd: chown
applmgr:dba /oraapplx/applmgr/product/1Ø7/sch/1.Ø/log/ijnp*.txt started
```

Example 2 : ftp received all files but there is no path and no action defined in ftp.ini

```
15/Ø9/99 1Ø:55:ØØ 937ØØ root   /home/oper/ftp_receive - ftp - files_order.cmd: Path
not found in ftp.ini
15/Ø9/99 1Ø:55:Ø1 937ØØ root   /home/oper/ftp_receive - ftp - files_order.cmd: No
action defined in ftp.ini
```

Example 3 : error in ftp.ini

```
28/Ø5/99 1Ø:35:Ø2 root   ftp_receive - ftp - bls_ecodet_R1.cmd: Path not found in
ftp.ini
28/Ø5/99 1Ø:35:Ø2 root   ftp_receive - ftp -
****************************************************
28/Ø5/99 1Ø:35:Ø3 root   ftp_receive - ftp - *
*
28/Ø5/99 1Ø:35:Ø3 root   ftp_receive - ftp - * bls_ecodet_R1.cmd: error in file(s)/
ftp.ini, chown_to_ applmgr NOT started *
28/Ø5/99 1Ø:35:Ø3 root   ftp_receive - ftp - *        call standby
*
28/Ø5/99 1Ø:35:Ø3 root   ftp_receive - ftp - *         Script halted
*
28/Ø5/99 1Ø:35:Ø4 root   ftp_receive - ftp - *
*
28/Ø5/99 1Ø:35:Ø4 root   ftp_receive - ftp -
****************************************************
```

ftp_send

Example 1 : file does not exist --> ftp does not take place

```
15/Ø9/99 1Ø:4Ø:Ø1 539Ø4 ftp_user  ftp_send - ftp -
****************************************************
15/Ø9/99 1Ø:4Ø:Ø1 539Ø4 ftp_user  ftp_send - ftp - *
*
```

```
15/Ø9/99 1Ø:4Ø:Ø1 539Ø4 ftp_user  ftp_send - ftp - * file_order.cmd: File /ftp/send/
order_12 not present *
15/Ø9/99 1Ø:4Ø:Ø1 539Ø4 ftp_user  ftp_send - ftp - *          call standby
*
15/Ø9/99 1Ø:4Ø:Ø1 539Ø4 ftp_user  ftp_send - ftp - *
*
15/Ø9/99 1Ø:4Ø:Ø1 539Ø4 ftp_user  ftp_send - ftp -
****************************************************
```

Example 2 : All files exists and ftp went OK

```
15/Ø9/99 14:3Ø:Ø1 8913Ø ftp_user  /home/oper/ftp_send - ftp - file_pickera: ftp to
picker_server is okay
15/Ø9/99 14:3Ø:Ø3 8913Ø ftp_user  /home/oper/ftp_send - ftp - file_pickerd: ftp to
picker_server is okay
15/Ø9/99 15:4Ø:Ø2 99Ø7Ø ftp_user  /home/oper/ftp_send - ftp - file_pickerf: ftp to
picker_server is okay
15/Ø9/99 15:4Ø:Ø4 99Ø7Ø ftp_user  /home/oper/ftp_send - ftp - file_pickerj: ftp to
picker_server is okay
15/Ø9/99 15:4Ø:Ø5 99Ø7Ø ftp_user  /home/oper/ftp_send - ftp - file_picker.cmd: FTP to
picker_server is okay
15/Ø9/99 15:4Ø:Ø5 99Ø7Ø ftp_user  /home/oper/ftp_send - ftp - file_picker.cmd: all
files to picker_server transferred
```

Example 3 : FTP of the previous session is still busy, script deletes lockfile

```
16/Ø9/99 19:47:41 27488 ftp_user  /home/oper/ftp_send      ------- Lock found in
ftp_send
16/Ø9/99 19:57:41 31466 ftp_user  /home/oper/ftp_send      ------- Lock found in
ftp_send
16/Ø9/99 2Ø:Ø7:41 23326 ftp_user  /home/oper/ftp_send      ------- Lock found in
ftp_send
16/Ø9/99 2Ø:Ø7:41 26432 ftp_user  /home/oper/ftp_send      ------- Ftp is locked 3Ø
minutes
16/Ø9/99 2Ø:Ø7:41 26432 ftp_user  /home/oper/ftp_send      ------- Checking ftp in
ps -ef
16/Ø9/99 2Ø:Ø7:41 26432 ftp_user  /home/oper/ftp_send      ------- No ftp in ps -
ef, removing lockfile
```

Example 4 : FTP of the previous sessions is still busy, user has to check ftp

```
19/Ø9/99 19:49:41 17468 ftp_user  /home/oper/ftp_send      ------- Lock found in
ftp_send
19/Ø9/99 19:59:41 17472 ftp_user  /home/oper/ftp_send      ------- Lock found in
ftp_send
19/Ø9/99 2Ø:Ø9:41 1748Ø ftp_user  /home/oper/ftp_send      ------- Lock found in
ftp_send
```

```
19/Ø9/99 2Ø:Ø9:45 1748Ø ftp_user /home/oper/ftp_send     ------- Ftp is locked 3Ø
minutes
19/Ø9/99 2Ø:Ø9:45 1748Ø ftp_user /home/oper/ftp_send     ------- Checking ftp in
ps -ef
19/Ø9/99 2Ø:Ø9:45 1748Ø ftp_user /home/oper/ftp_send     ------- Ftp still busy,
giving
it 1Ø minutes more before serious warning
19/Ø9/99 2Ø:Ø9:45 1748Ø ftp_user /home/oper/ftp_send     ------- Lock found in
ftp_send
19/Ø9/99 2Ø:1Ø:3Ø 1751Ø ftp_user /home/oper/ftp_send     ------- Ftp is locked 4Ø
minutes
19/Ø9/99 2Ø:1Ø:3Ø 1751Ø ftp_user /home/oper/ftp_send     ------- Checking ftp in
ps -ef
19/Ø9/99 2Ø:1Ø:3Ø 14888 ftp_user /home/oper/ftp_send     - ftp -
***********************
****************************
19/Ø9/99 2Ø:1Ø:31 14888 ftp_user /home/oper/ftp_send     - ftp - *
                           *
19/Ø9/99 2Ø:1Ø:31 14888 ftp_user /home/oper/ftp_send     - ftp - *  Check ftp:
ftp session 4Ø minutes busy    *
19/Ø9/99 2Ø:1Ø:31 14888 ftp_user /home/oper/ftp_send     - ftp - *
call standby                   *
19/Ø9/99 2Ø:1Ø:31 14888 ftp_user /home/oper/ftp_send     - ftp - *
                           *
19/Ø9/99 2Ø:1Ø:31 14888 ftp_user /home/oper/ftp_send     - ftp -
***********************
****************************
19/Ø9/99 2Ø:1Ø:32 1751Ø ftp_user /home/oper/ftp_send     ------- Lock found in
ftp_send
```

ftp_cleanup

```
15/Ø9/99 Ø1:45:55 538Ø8 root  /home/oper/ftp_cleanup ------- Begin
15/Ø9/99 Ø1:47:15 538Ø8 root  /home/oper/ftp_cleanup ------- End

entries in /home/data/ftp_removed:
/ftp/send/backup/order_1.cmd
/ftp/send/backup/order_1.cmd.ftp
/ftp/send/backup/picker_2
/ftp/send/backup/picker_12
/ftp/send/backup/picker_17
/ftp/send/backup/picker_19
/ftp/receive/backup/file_Ø1.dmp
/ftp/receive/backup/file_Ø8.dmp
/ftp/receive/backup/file12.dmp
/ftp/receive/backup/file19.dmp
/ftp/receive/backup/file22.dmp
/ftp/receive/backup/file24.dmp
/ftp/receive/backup/wms_11.cmd
```

```
/ftp/receive/backup/wms_14.cmd
/ftp/receive/error/bls_ecodet_1.cmd
/ftp/receive/error/bls_ecodet_R1.cmd
```

ftp_restart

example1 : ftp_restart started on the prompt

```
The following ftps have been stopped due to errors:
Nr Command-file    Date
1 : testftp1.cmd Ø7/Ø8 1Ø:5Ø:ØØ
2 : testftp2.cmd Ø7/Ø8 1Ø:4Ø:ØØ
3 : testftp3.cmd Ø6/Ø8 1Ø:4Ø:ØØ

Ø : Quit

Which FTP can be restarted ? 2


The following files have been transferred the previous time:
ØØ1893COØ1.NEW

 Send again? y/N: n

The files mentioned above will not been transferred again.

The following files will be transferred to server1.domain3.company
ØØØØØØØOØ1.OLD


FTP of testftp2.cmd processing for restart


*
*  FTP of testftp2.cmd is ready to restart
*  FTP of testftp2.cmd will be processed in the next loop of ftp_send
*
```

example2 : logging ftp_restart started in the background

```
Ø6/Ø9/Ø1 15:Ø3:Ø8 119942 ftp_user  ftp_restart   ------- Processing testftp1.cmd
Ø6/Ø9/Ø1 15:Ø3:1Ø 119942 ftp_user  ftp_restart   ------- The following files have
already been transferred and will not be transferred:
Ø6/Ø9/Ø1 15:Ø3:11 119942 ftp_user  ftp_restart   -------
Ø6/Ø9/Ø1 15:Ø3:11 119942 ftp_user  ftp_restart   ------- The following files will be
transferred to server2.domain1.company
Ø6/Ø9/Ø1 15:Ø3:11 119942 ftp_user  ftp_restart   ------- PROD.UCTION.FILE1
Ø6/Ø9/Ø1 15:Ø3:12 119942 ftp_user  ftp_restart   ------- Processing testftp2.cmd
Ø6/Ø9/Ø1 15:Ø3:14 119942 ftp_user  ftp_restart   ------- The following files have
already been transferred and will not be transferred:
```

```
Ø6/Ø9/Ø1 15:Ø3:15 119942 ftp_user ftp_restart   ------- ØØ1893COØ1.NEW
Ø6/Ø9/Ø1 15:Ø3:15 119942 ftp_user ftp_restart   ------- The following files will be
transferred to server1.domain3.company
Ø6/Ø9/Ø1 15:Ø3:15 119942 ftp_user ftp_restart   ------- ØØØØØØØØOØ1.OLD
Ø6/Ø9/Ø1 15:Ø3:16 119942 ftp_user ftp_restart   ------- Processing testftp3.cmd
Ø6/Ø9/Ø1 15:Ø3:18 119942 ftp_user ftp_restart   ------- The following files have
already been transferred and will not be transferred:
Ø6/Ø9/Ø1 15:Ø3:19 119942 ftp_user ftp_restart   ------- ØØØ136COØ8.NEW
Ø6/Ø9/Ø1 15:Ø3:19 119942 ftp_user ftp_restart   ------- The following files will be
transferred to server6.domain2.company.nl
06/09/01 15:03:19 119942 ftp_user ftp_restart   ------- There are no files to
send, files will be moved to /ftp/send/backup
```

The code for this article is available for download from our Web site, at www.xephon.com/extras/autoftp.txt.

Teun Post, Schuitema N.V.
(The Netherlands)

March 1997 – March 2004 index

Items below are references to articles that have appeared in *TCP/SNA Update* since March 1997. References show the issue number followed by the page number(s). All these back-issues of *TCP/SNA Update* can be ordered from Xephon. See page 2 for details.

TCP/SNA news

Websense, the provider of employee Internet management (EIM) solutions, has announced that its content filtering solution is now available on Nokia's purpose-built IP security appliances.

The combination enables businesses to detect and manage employee access to the Internet and other network-based applications, such as instant messaging, peer-to-peer (P2P) file-sharing, and streaming media.

Features include:
- A large database of Web sites.
- Advanced filtering for pop-up ads, Internet radio and TV, spyware, and mobile malicious code.
- Flexible filtering options, including time-based quotas, warn/continue, and password authentication.
- Dynamic network protocol management for instant messaging, P2P, and streaming media.
- Support for real-time and trend reporting tools such as Websense Reporter, Websense Explorer, and Websense Real-time Analyzer.

Websense has also announced that Websense Enterprise, along with its desktop security solution Client Application Manager (CAM), can stop blended threats like W32/Mydoom and other similar worms.

URL: http://www.websense.com/nokia

* * *