49 TCP/SNA update

March 2003

In this issue

TCP/SNA Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38342
From USA: 01144 1635 38342
E-mail: fionah@xephon.com

North American office

Xephon
Post Office Box 350100
Westminster, CO 80035-0100
USA
Telephone: (303) 410-9344

Subscriptions and back-issues

A year's subscription to *TCP/SNA Update*, comprising four quarterly issues, costs £130.00 in the UK; $190.00 in the USA and Canada; £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 1999 issue, are available separately to subscribers for £33.00 ($49.50) each including postage.

Editorial panel

Articles published in TCP/SNA Update are reviewed by our panel of experts. Members include John Bradley (UK), Carlson Colomb (Canada), Anura Gurugé (USA), Jon Pearkins (Canada), and Tod Yampel (USA).

Editor

Fiona Hewitt

Contributions

When Xephon is given copyright, articles published in *TCP/SNA Update* are paid for at £170 ($260) per 1000 words for original material. To find out more about contributing an article, please download a copy of our *Notes for Contributors* from http://www.xephon.com/index/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..

Printed in England.

Getting ready to implement Web services

Web services shouldn't be summarily dismissed as mere marketing hype just because they have yet to live up to the enormous expectations that have surrounded them since their inception over two years ago.

There's no doubt that Web services have yet to gain any real traction in real-life production use, despite the near-unprecedented push they've received from every quarter of the computer industry. It's also true that 2002, which many expected to be the year that Web services would begin to flourish, saw only a staggering amount of effort being expended by the supply side (ie the vendor community) on defining and publicizing Web services-related standards – particularly in the security and business process 'orchestration' fronts – rather than any real proliferation of mission-critical Web services.

Despite these very public setbacks, however, Web services are very real, viable, and here to stay – for a very long time to come. Later, we'll examine some of the reasons why Web services have been slow to live up to their expectations; first, however, let's review just why Web services are so important to the SNA community, and to those associated with application development, application support, or system support in particular.

Over the next few years, much of the proven, highly valuable, business-critical functionality that is still embodied within decades-old SNA applications will need to be converted into Web services. So, to paraphrase that adage about old soldiers, we can now say, without any fear of contradiction, that: "Old SNA applications never die, they just fade away to become Web services."

If you think of Web services in terms of protocols such as SOAP, WSDL, and UDDI, it may not be obvious what the connection is to the future of mission-critical SNA applications. So it's worth taking a moment to do some fundamental level-setting.

AN APPLICATION BY ANY OTHER NAME

Web services are modular, self-contained 'applications'. Despite the confusion in the media and even in some White Papers and technical documentation from companies that should know better, Web services in themselves are not abstract middleware services related to locating, defining, and invoking applications. Web services are meant to be functionally complete, task-performing software components. That is a given, and should no longer be subject to doubt or debate.

Hence the tie-in with existing SNA applications. A Web service is a remotely invokable unit of software. In today's business world, mission-critical SNA applications contain some of the most proven and most valuable units of business-logic-related software, including software for customer credit rate calculations, customer account maintenance, portfolio management, manufacturing process automation, and customer account look-up.

IBM understands the role that Web services can play when it comes to extending the already healthy ROI of 'legacy' applications – one of the key features now cited for IBM WebSphere Host Publisher is the ability to enable existing legacy services (resident within SNA mission-critical applications) to be deployed as new Web services.

Since IBM's perspective with WebSphere Host Publisher is 100% SNA-centric, it's an excellent place to start in your quest to understand the potential of Web services. Host Publisher's SNA-related Web services capabilities are complemented and augmented by the Web services support now available in IBM's WebSphere Application Server Version 5 and the new revamped WebSphere Studio. Your first step, therefore, should be to visit IBM's portal and check out what IBM is offering in terms of Web services-enabling technology within the WebSphere family.

The goal of Web services is to enable you to develop new Web applications, relatively quickly, that consist of software components from multiple, diverse sources (including those extracted from existing SNA applications) that have been

dynamically assembled and loosely coupled together. This is, essentially, distributed computing on the Web, with various applications – many in the form of Web services – dynamically interacting with each other. A Web service, however, is typically not meant to be a full-blown, feature-rich application in its own right (though, to be fair, there are no actual restrictions as to how long, big, or complex a Web service can or should be).

For a start, a Web service doesn't necessarily have to possess a user interface (though IBM has recently been promoting a type of Web service, in particular for portal applications, that comes complete with its own GUI). However, the idea of having a Web service without a user interface runs counter to most people's notion of what constitutes an application. So, although it needs to be characterized as an application for technical integrity, it's better to think of a Web service as a 'mini-application' or even an application 'segment'. This means that, although an existing mission-critical SNA application could be made into a single Web service, it's unlikely to be packaged in this way. Instead, what's likely to happen is that various parts of it – in particular business-logic-related functionality – will be isolated and made into discrete Web services.

There are a number of other terms that could be used to convey the true essence of a Web service; you may prefer one or other of these, depending on your background: subroutine, software building block, reusable object, software component, chunk of business logic, entry (or member) from a software library, remote procedure (or even possibly remote procedure call), or business process representation.

SNA WILL BE A DONOR RATHER THAN A CONSUMER

If you're currently responsible for sustaining one or more SNA applications, it's highly likely that you'll soon be asked by management to make some recommendations as to how your application or applications 'relate' to Web services. In general, this would be a 'fishing expedition'. Corporate IT, around the globe, is still trying to work out the implications and ramifications

of all this fuss about Web services. The confusion factor gets even greater when ‘legacy’ is thrown into the mix.

The key issue that needs to be worked out here is whether SNA applications are going to be Web service consumers or Web service sources (or providers).

An application that relies on software functionality provided by a Web service can be thought of as a Web service consumer. Although there’s no technical reason why existing SNA applications shouldn’t become Web service consumers, this is unlikely to happen in practice. An SNA application will need to become a Web service consumer only if you’re thinking of extending its current functionality to include capabilities best obtained in the form of Web services. So this boils down to the economics and practicality of modifying and extending applications that in many cases were developed even before the advent of the PC. Rather than going and modifying the code of these applications, most of which were written in COBOL, BAL, or PL/I, a better way to extend their functionality would be in terms of host integration (see below). The bottom line here, however, is that, post Y2K, most IT organizations are reluctant to make any more large-scale extensions to decades-old SNA applications. This is why SNA applications are unlikely to be major consumers of Web services, other than from a host integration perspective.

On the other hand, SNA applications are veritable smorgasbords when it comes to relevant business logic that can be packaged for re-use as Web services. This also ties in with the very hot and strategic interest in business process integration. The lifeblood of any enterprise is its processes; it is they that sustain the business. In the case of an insurance company, for example, there will be multiple processes that relate to the issuing, updating, and cancelling of customer policies. Each such business process typically involves the invocation of multiple applications. Business process integration as it applies to enterprise applications is all about streamlining this mechanism so that all the necessary applications can be seamlessly integrated.

SNA applications embody umpteen proven, critical, and hard-to-accurately-reproduce business processes. With Web services, these business processes can be isolated, neatly packaged, and readily marketed for use by new in-house applications being developed – or even by other entities. And, because Web services are totally platform-independent, there are no real issues about whether one person's application will be able to use another's Web service. A new Web application being developed in Visual Basic to run on a Microsoft .NET platform can transparently make use of business process Web services derived from SNA applications running on a mainframe.

THE FIRST CHALLENGE FOR SNA 'CURATORS'

The first challenge that you face when it comes to Web services is to identify the individual business processes embodied within your SNA applications. You'll effectively have to do an audit of the applications and catalog the various business-related functionality offered by each application. Some of this application 'audit' groundwork may have been done as a part of the Y2K extravaganza, in which case you can use that as a starting point and build a catalog of the software functionality already 'in-stock' within your data centre.

The next step is to determine how practical it would be to isolate each of the business logic units identified in the above step. This is where you need to look at SNA-specific host integration tools such as IBM's WebSphere Host Publisher, NetManage's OnWeb, and SEAGULL's Transidiom. These provide mechanisms to isolate business logic components from within SNA applications, with IBM's Host Publisher (right now at least) offering the greatest tie-in with Web services protocols. Once you're comfortable that you can extract functionality from SNA applications and make them into Web services, using tools such as Host Publisher, you can report back to management on your recommendation with regard to Web services and SNA applications.

Typically, you would show management your catalog of potential Web services that can be derived from the SNA applications.

Then, politics, avarice, competitiveness, self-preservation, and other related factors will come into play. This is a new ball game for most enterprises and people. Suddenly, you have a standards-based mechanism whereby you can market and share software functionality that has been closely guarded in-house for decades. In many cases, the status quo will prevail; management will, understandably, not be ready to rush off and start offering company software resources as Web services unless they really have to.

It's at this point that pricing and security issues arise. The pricing model for Web services is still in its embryonic stage; most people, quite rightly, are still focusing their attention on the technical side of things. To its credit, however, IBM has already started publishing position papers about possible accounting and metering schemes for Web services. What's clear is that there will be a very wide spectrum, ranging from freeware to expensive premium offerings, and many permutations as to how the pricing will be structured. The pricing options available will definitely include one-time charge schemes, periodic licensing (eg monthly or yearly), and umpteen usage-based options. In addition, there will probably also be third-party Web service distributors, although the inherent dynamic discovery capability of a Web service dilutes some of the potential value that can be offered by a distributor. In the case of Web services, the main value of a distributor will be in handling the billing and collection. There will doubtless also be syndicated Web services.

Management will no doubt want to table the whole issue of Web service provision – now that they know that it can be done – until the pricing model becomes more formalized.

WEB SERVICES AND HOST INTEGRATION

Since at least 1999, host integration has been positioned as the final frontier of Web-to-host. Web-to-host via applet-based emulation (eg IBM's Host On-Demand) or 3270/5250-to-HTML conversion (eg Hummingbird's e-Gateway) enabled cost-effective, Web-based, Web browser-invoked access to existing

SNA applications running on mainframes or AS/400s. Host integration went one step further, and talked about how existing SNA applications could complement new Web applications; it was all about reusing the software functionality found within SNA applications. Web services provides the ideal mechanism for this software reuse scenario. Hence the now close interplay between SNA applications, host integration, and Web services.

Host integration is an 'execution-time' software binding mechanism – the software is being reused rather than 'cut-and-pasted' from the old application to the new, it continues to run on a host platform, and interacts with the new application in terms of dynamic I/O calls. This is also the Web service model. What's more, Web services provide all of this within a highly standards-based, XML-centric framework which is platform- and programming language-independent.

With host integration (eg with IBM's Host Publisher), the functionality of SNA applications can be significantly enhanced without any changes to the source code of the original application. Rather than adding new code to the existing application, you can create new code that runs as a separate process and interacts with the original application at run time. The new code can be in the form of Web services. Any consumption of Web services by SNA applications is therefore best viewed in the context of host integration, as opposed to SNA applications being modified to locate and invoke Web services.

SOME EXPLANATIONS AS TO WHY WEB SERVICES ARE LATE

Web services aren't a solution or 'killer application' in their own right. Rather, they're a support or infrastructure mechanism for new application development. This distinction by itself can explain many of the set-backs experienced by Web services. Since 2000, thanks to the recessionary pressures being felt around the globe, enterprises have been very wary of undertaking costly new software development projects. Because there aren't that many applications being developed right now, there isn't as much demand for Web services as we might like. It's a kind of chicken-and-egg scenario: application development

needs to take off again in order for Web services to prove their worth.

This said, it's true that the industry started hyping Web services in autumn 2000 before they were truly ready for prime time! It wasn't difficult for technical people to see the potential and promise of Web services: they addressed, in one fell swoop, so many issues that in the past had hindered software reuse. Moreover, Web services were totally Web centric. After the humbling of the dot-coms, Web services promised to be the next big thing – hence the rush to market.

Although the key enabling protocols – SOAP, WSDL, and UDDI – were in place, however, many of the other infrastructure-related standards, especially in the areas of security and business process 'orchestration', were missing. Fortunately, however, 2002 saw tremendous progress in the areas of security and orchestration, and these are no longer pitfalls. The pricing model is the next big thing that needs to be nailed down.

SECURITY AS IT APPLIES TO WEB SERVICES

As reusable pieces of business logic, sourced and invoked over the Web, Web services have the potential to be the ultimate in 'Trojan horse' type threats – even taking into account that the WS code is executed, remotely, on a third-party server. The first and biggest concern here is making sure that the provider of a Web service is really who they claim to be – especially if you intend to be dealing with sensitive data.

Then there's the whole issue as to what happens to the information you've shared with a Web service. What rights, explicit and implicit, does a provider of a Web service have when it comes to storing, analysing, and, above all, exploiting (eg selling) what's been sent to the Web service?

This clearly opens up a whole Pandora's box of issues. Today we worry about the information that portals (eg Amazon) automatically and transparently extract from visitors via what's referred to, euphemistically, as 'collaborative filtering'. The

scope for this type of intrusion in the case of Web services is equally high – and more insidious. There's a clear need, therefore, to determine that a WS is not unscrupulous; and the problem here is that scruples come in many shades of grey.

Although a major concern, securing the privacy of the information flowing to and from a WS is not a real problem. Proven solutions like SSL-based encryption will work, and are more than adequate for the time being.

Unfortunately, however, there are other problems. Think of the various 'buffer overflow' type vulnerabilities that have been exploited in Windows NT and some Web servers. Well, WSs, at least in theory, can take such threats to a whole new dimension. Related to this is the possibility of denial of service attacks propagated through the WS connections – especially because SOAP-based WS traffic is typically configured to flow through ports 80 and 443 – standard HTTP and SSL-encrypted traffic flows for Web servers, respectively. All in all, it's not surprising that people want to securely nail down the WS security issues.

There is now even a forum, the XML Web Services Security (XWSS), at www.xwss.org to act as a central clearing house for WS-related security issues. It's definitely worth visiting this forum to familiarize yourself with what's happening in this fast-evolving arena.

The WS security-related issues being addressed by the XWSS include the following:

- The issue of service provider and service requester authentication. This is a mandatory prerequisite for any service involving sensitive information. Authentication, as we saw in the context of corporate portals last issue, can be realized quite successfully using multiple different highly-proven technologies. In the context of Web services, you could also think about using Kerberos and LDAP.
- Authorization and access control. This, in effect, is the next layer of security after bi-directional authentication has been

performed. Authorization, which is typically implemented using user-ids and passwords, determines the functions or resources to which a particular user has rightful access. Rules-based access control mechanisms can be employed here, depending on the level and sensitivity of the functions being offered by a Web service.

- Single sign-on is another factor when it comes to authorization and access control, if multiple Web services are available from the same provider. A methodology called 'security access mark-up language' (SAML), is being developed in an attempt to represent relevant authentication and authorization parameters and criteria in standard XML form.
- Data encryption between the data flowing between a WS requester and provider – which would, as previously mentioned, be typically handled using standard SSL encryption along the lines of HTTPS.
- Digital signatures and non-repudiation. When dealing with sensitive data across the Internet, now in the context of Web services, it's imperative to have a mechanism to ensure total data integrity – both to ensure that the data wasn't tampered with while it was crossing the Internet, and to prevent the WS from refusing to perform the transaction (the non-repudiation aspect). This is where digital signatures will come into play.

As well as addressing the above issues in the context of Web services, the XWSS is also already making strides on the notion of 'XML Application Firewalls' – particularly as they apply to Web services. It should be fairly obvious what this type of firewall is trying to achieve. It focuses on SOAP-borne messages flowing to and from Web services, and tries to determine that they are authorized and unmalicious.

XML KEY MANAGEMENT (XKMS)

XML key management (XKMS), developed by folks at Microsoft, VeriSign, and webMethods, sets out to integrate Public Key

Infrastructure (PKI) with the Internet backbone using XML methodologies. Its goal is to specify protocols for distributing and registering public keys as used in dual-key public-key cryptography.

XKMS is designed for use in conjunction with the proposed standard for XML Signature (XML-SIG) developed by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF) as well as an anticipated companion standard for XML encryption.

PKI, as indicated by the inclusion of the term 'infrastructure' in its name, provides a framework for facilitating secure – but at the same time accessible and easy-to-use – public-key encryption and digital signatures. The term 'public' in the title, however, can be deceptive. PKI is not a centralized, public service for distributing and managing keys. The 'public' in this context refers to the public-key in the public-private key combination. Today, PKIs are typically implemented on a corporate basis using PKI facilitating products from a variety of vendors including IBM, VeriSign, and Entrust. IBM now even includes a comprehensive PKI on its flagship z/OS operating system.

There is a three-way relationship between PKI, digital certificates, and digital signatures. The use of digital certificates (DCs) for Internet applications is dependent on PKI; without PKI, DCs would not be as convenient to use or administer. PKI has four main components:

- Certificate Authorities (CAs), eg VeriSign, that issue and validate digital certificates.
- One or more registration authorities (RAs) that work with the CA to verify a requester's identity before the digital certificate is issued.
- One or more public directories (typically based on LDAP) where digital certificates, which will contain the public keys of their owners, can be stored.

- A comprehensive and foolproof certificate management system.

As an infrastructure scheme, PKI is expected to cater for key back-up and recovery. Users typically have to enter a password to access their public-key encryption keys. But users forget their password or may leave the company. If there's no way to recover encryption/decryption keys in such instances, valuable company information that has previously been encrypted may be lost forever.

It's therefore essential to have a secure and reliable way for a designated and fully-validated security administrator to recover encryption key pairs through the PKI system. Since such recovery will be possible only if the key pairs are properly backed up, it's assumed that the CA will automatically keep a back-up of keys that have been issued.

XKMS's goal is to make PKI into a formalized, centralized, 'public' service as connoted by the term. XKMS consists of two parts:

- XML Key Information Service Specification (X-KISS)
- XML Key Registration Service Specification (X-KRSS).

X-KISS defines a protocol for a Trust Service. This trust service will resolve public key information contained in 'XML-SIGelements'. The X-KISS protocol allows a client of such a service to delegate part or all of the tasks required to process such elements.

A key objective of X-KISS is to minimize the complexity of application implementations by allowing them to become clients and thereby to be shielded from the complexity and syntax of the underlying PKI used to establish trust relationships. The underlying PKI may be based on a different specification such as X.509.

X-KRSS specification defines a protocol for a Web service that accepts registration of public key information. Once registered, the public key can be used in conjunction with other WSs, including X-KISS.

Both protocols, as expected, are defined in terms of structures expressed in terms of XML schema and conveyed using SOAP. The relationships among the messages used are defined using WSDL.

Web Services Security (WS-Security), developed by folks from IBM, Microsoft, and VeriSign, defines enhancements to SOAP to provide quality of protection for SOAP-borne messages through message integrity, message confidentiality, and single message authentication – with regard to the concerns already raised at XWSS.org. WS-Security is meant to be used by WSs to realize standards-based data and service integrity and confidentiality.

WS-Security is defined in terms of the Web Services Security Language, which gets abbreviated and referred to as WS-Security. WS-Security is an 'open' and extensible mechanism that can be used to accommodate a wide variety of security models and encryption technologies, including PKI, Kerberos, and SSL/TLS. WS-Security provides support for multiple security tokens, multiple trust domains, multiple signature formats, and multiple encryption technologies.

WS-Security, as shown below, includes a general-purpose mechanism for associating security tokens with SOAP messages. It doesn't call for a specific type of security token; it's designed to support multiple security token formats. For example, a client might provide proof of identity as well as further proof that they have a particular business certification.

WS-Security also describes how to encode binary security tokens – in particular X.509 certificates and Kerberos tickets. It also includes mechanisms that can be used to further describe the characteristics of the credentials that are included with a SOAP message.

WS-Security provides for three main mechanisms:

- Security token propagation
- Message integrity
- Message confidentiality.

BOTTOM LINE

Although they haven't materialized as quickly as had been originally forecast, Web services are real, viable, and here to stay. What's more, they're particularly relevant to the SNA world, since they provide the mechanism whereby the invaluable business logic contained within existing mission-critical applications can be repackaged for reuse by future generations of application developers. One of the key reasons why Web services have been slow to take off was a lack of security measures, but this has now been rectified. If you're an SNA 'curator' you should start looking at how to convert your SNA applications to Web services using the pointers contained in this article.

Anura Gurugé
Strategic Consultant (USA)

A TCP/IP transaction for linking CICS and PC programs

Creating and testing programs for TCP/IP communication is on-going and hard work. You can't use CEDF, and CEDX is not much use either. This article presents two universal modules – TCP1 on IBM and TCPACTIVEX on a PC – which can be used to simplify TCP/IP communication between CICS and PCs using sockets. The modules are easy to use, and can be used from a variety of PC programs and for calling a CICS program on IBM.

TCPACTIVEX is a Visual Basic ActiveX.dll that links a CICS program with PC programs using the RunCICS function (see Figure 1). This function has the following parameters:

- *PID*. Program ID of the CICS program.
- *CICSInput*. Input for the requested CICS program.

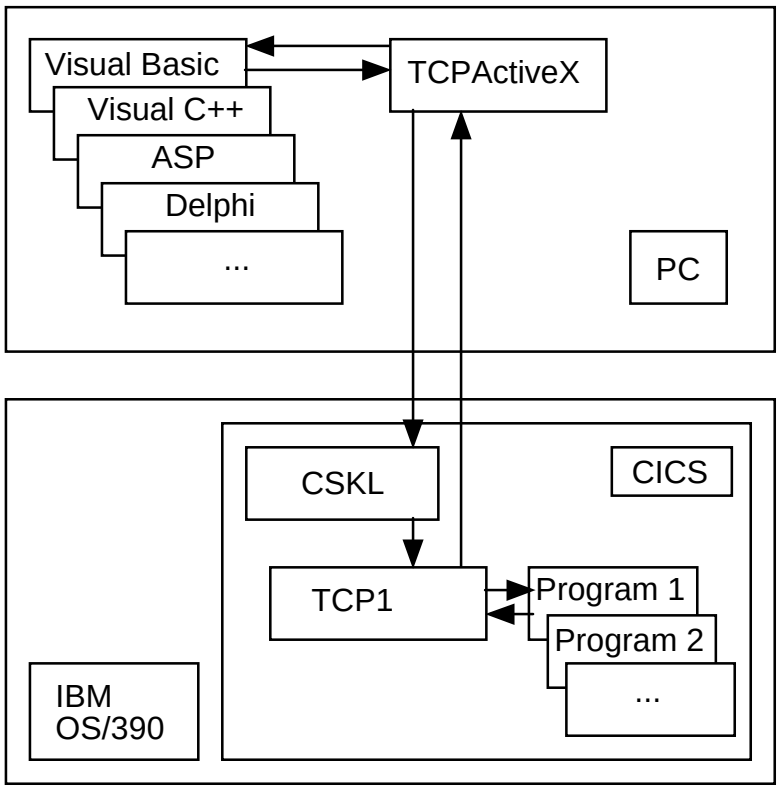

Figure 1: Communication between a PC and CICS

- *CICSOutput*. The common area after execution, ie the results.
- *ErrDesc*. If the function returns a non-zero value, this is a description of the error.

TEST1

Below is a sample VB program for testing TCPACTIVEX.

```
Private Sub Command1_Click()
Dim obj As New TCPACTIVEX.TCPClass1
Dim res As Integer
Dim CICSRes As String
Dim ed As String
res = obj.RunCICS("Ø1", "12340000", CICSRes, ed)
If res <> Ø Then
    MsgBox Trim(ed) & " (CODE " & res & ")"
Else
    MsgBox CICSRes
End If
End Sub
```

TCP1 is a CICS transaction that is called from CSKL. TCP1 has the following functions:

- Receiving the RunCICS parameter CICSInput from TCP/IP.
- Processing the link to the requested CICS program (with the received data as input).
- Sending the results as the RunCICS parameter CICSOutput via TCP/IP.

TCPIP01

```
TCPIPØ1: PROC OPTIONS(MAIN);
/*******************************************************/
/* TCP1 - TRANSACTION                                  */
/* FOR LINKING CICS PROGRAMS AND PC                    */
/* VIA TCP/IP                                          */
/*                                                     */
/* DEJAN JELIC                          APR 2ØØ2.      */
/*******************************************************/

DCL EZASOKET ENTRY OPTIONS(ASSEMBLER,RETCODE) EXTERNAL;
/* FROM EBCDIC TO ASCII FOR SENDING DATA*/
DCL EZACICØ4 ENTRY OPTIONS(ASSEMBLER,RETCODE) EXTERNAL;
/* FROM ASCII TO EBCDIC FOR RECEIVING DATA*/
DCL EZACICØ5 ENTRY OPTIONS(ASSEMBLER,RETCODE) EXTERNAL;

DCL (SUBSTR,ADDR,CSTG,STG,LENGTH,VERIFY) BUILTIN;

DCL TASK_START CHAR(5)   INIT('#S~S#');

DCL 1  SOKET_FUNCTIONS,
      2 SOKET_CLOSE          CHAR(16) INIT('CLOSE          '),
      2 SOKET_RECV           CHAR(16) INIT('RECV           '),
      2 SOKET_TAKESOCKET     CHAR(16) INIT('TAKESOCKET     '),
      2 SOKET_WRITE          CHAR(16) INIT('WRITE          ');
DCL TOEBCDIC_TOKEN CHAR(16)      INIT('TCPIPTOEBCDICXLT');
DCL TOASCII_TOKEN  CHAR(16)      INIT('TCPIPTOASCIIXLAT');
DCL TAKE_SOCKET    BIN FIXED(31) INIT(Ø);
DCL SOCKID         BIN FIXED(15) INIT(Ø);
DCL SOCKID_FWD     BIN FIXED(31) INIT(Ø);
DCL ERRNO          BIN FIXED(31) INIT(Ø);
DCL AF_INET        BIN FIXED(31) INIT(2);

DCL 1 TCPBUF,
      2 PROGID CHAR(2) INIT(''),
```

```
      2 DATA CHAR(500)  INIT('');
DCL TCPBUFLEN BIN FIXED(31) INIT(0);

DCL RECV_FLAG BIN FIXED(31) INIT(0);
DCL CLENG     BIN FIXED(15) INIT(0);

DCL 1 CLIENTID_LSTN,
      2 CID_DOMAIN_LSTN      BIN FIXED(31) INIT(0),
      2 CID_NAME_LSTN        CHAR(8)       INIT(' '),
      2 CID_SUBTASKNAME_LSTN CHAR(8)       INIT(' '),
      2 CID_RES_LSTN         CHAR(20)      INIT(' ');
DCL CLIENTID_LSTN_BAZ CHAR(40) BASED(ADDR(CLIENTID_LSTN));

DCL 1  CLIENTID_APPL,
      2 CID_DOMAIN_APPL      BIN FIXED(31) INIT(0),
      2 CID_NAME_APPL        CHAR(8)       INIT(' '),
      2 CID_SUBTASKNAME_APPL CHAR(8)       INIT(' '),
      2 CID_RES_APPL         CHAR(20)      INIT(' ');

DCL 1 TCPSOCKET_PARM,
      2 GIVE_TAKE_SOCKET  BIN FIXED(31) INIT(0),
      2 LSTN_NAME         CHAR(8)       INIT(' '),
      2 LSTN_SUBTASKNAME  CHAR(8)       INIT(' '),
      2 CLIENT_IN_DATA    CHAR(35)      INIT(' '),
      2 FILLER            CHAR(1)       INIT(' '),
      2 SOCKADDR_IN,
        3  SIN_FAMILY  BIN FIXED(15) INIT(0),
        3  SIN_PORT    BIN FIXED(15) INIT(0),
        3  SIN_ADDR    BIN FIXED(31) INIT(0),
        3  SIN_ZERO    CHAR(8)       INIT(' ');

DCL STATCICS BIN FIXED(31) INIT(0);
DCL TMODE BIT(1) INIT('0'B);
DCL MESSAGE1 CHAR(100) INIT('');
DCL STATCICSP PIC '(6)9' INIT(0);
DCL PRGSTAT BIN FIXED(31) INIT(0);

/************** P R O G R A M *************************/

PRGSTAT=INITIAL_SEC();
IF PRGSTAT=0 THEN PRGSTAT=TAKESOCKET_SEC();
IF PRGSTAT=0 THEN DO;
  TCPBUFLEN=PACKETR();
  IF TCPBUFLEN<0 THEN PRGSTAT=-1;
END;
IF PRGSTAT=0 THEN PRGSTAT=EXELINK();
DELAY(100);
IF TMODE THEN
  CALL EZASOKET(SOKET_CLOSE,SOCKID,ERRNO,STATCICS);
EXEC CICS RETURN;
```

```
INITIAL_SEC: PROC RETURNS(BIN FIXED(31));
   DCL RC BIN FIXED(31) INIT(Ø);
   CLENG=72;
   EXEC CICS RETRIEVE INTO(TCPSOCKET_PARM) LENGTH(CLENG)
             RESP(STATCICS);
   IF STATCICS¬=DFHRESP(NORMAL) THEN RC=-1;
   ELSE TMODE='1'B;
   RETURN(RC);
END INITIAL_SEC;

TAKESOCKET_SEC: PROC RETURNS(BIN FIXED(31));
   DCL RC BIN FIXED(31) INIT(Ø);
   CID_DOMAIN_LSTN=AF_INET;
   CID_DOMAIN_APPL=AF_INET;
   CID_NAME_LSTN=LSTN_NAME;
   CID_SUBTASKNAME_LSTN=LSTN_SUBTASKNAME;

   TAKE_SOCKET=GIVE_TAKE_SOCKET;
   SOCKID=GIVE_TAKE_SOCKET;
   SOCKID_FWD=GIVE_TAKE_SOCKET;
   CALL EZASOKET(SOKET_TAKESOCKET,SOCKID,
               CLIENTID_LSTN,ERRNO,STATCICS);
   IF STATCICS <  Ø  THEN RC=-1;
   ELSE DO;
     SOCKID=STATCICS;
     RC=PACKETS(TASK_START,5);
   END;
   RETURN(RC);
END TAKESOCKET_SEC;

PACKETS: PROC(POR,DUZ) RETURNS(BIN FIXED(31));
  /* SEND DATA */
  DCL RC BIN FIXED(31) INIT(Ø);
  DCL POR CHAR(51Ø);
  DCL DUZ BIN FIXED(31);
  CALL EZACIC04(TOASCII_TOKEN,POR,DUZ);
  CALL EZASOKET(SOKET_WRITE,SOCKID,DUZ,POR,ERRNO,STATCICS);
  IF STATCICS < Ø THEN RC=-1;
  RETURN(RC);
END PACKETS;

PACKETR: PROC RETURNS(BIN FIXED(31));
 /* RECEIVE DATA */
  TCPBUF=' ';
  TCPBUFLEN=5Ø2; /* MAX LENGTH */
  CALL EZASOKET(SOKET_RECV,SOCKID,
       RECV_FLAG,TCPBUFLEN,TCPBUF,ERRNO,STATCICS);
  /* STATCICS = NUMBER OF RECEIVED CHARACTERS */
  /* STATCICS < Ø FOR ERROR */
  IF STATCICS >= Ø THEN
```

```
     CALL EZACICØ5(TOEBCDIC_TOKEN,TCPBUF,STATCICS);
  RETURN (STATCICS);
END PACKETR;

SENDP: PROC(POD,DUZ) RETURNS(BIN FIXED(31));
  DCL POD CHAR(5ØØ);
  DCL DUZ BIN FIXED(31);
  DCL LPOD CHAR(51Ø);
  DCL LDUZ BIN FIXED(31);
  LPOD='#B~B#' || SUBSTR(POD,1,DUZ) || '#E~E#';
  LDUZ=DUZ+1Ø;
  RETURN(PACKETS(LPOD,LDUZ));
END SENDP;

EXELINK: PROC RETURNS(BIN FIXED(31));
 DCL RC BIN FIXED(31) INIT(Ø);
 DCL COMM CHAR(5ØØ);
 DCL PRGRECC CHAR(8Ø) INIT('');
 DCL 1 PRGREC BASED(ADDR(PRGRECC)),
       2 PRGID BIN FIXED(31),
       2 PRGNAME CHAR(8),
       2 PRGINPUL BIN FIXED(31),
       2 PRGOUTPL BIN FIXED(31),
       2 FILLER CHAR(6Ø);
 DCL KLJUC CHAR(4) BASED(ADDR(PRGID));

 IF VERIFY(TCPBUF.PROGID,'1234567890') ¬= Ø THEN
   RETURN(SENDP('ERRCICS:INVALID PARAMETER PROGID',32));

 PRGID=TCPBUF.PROGID;
 EXEC CICS READ DATASET('IØØ2VSAM') INTO(PRGRECC)
               RIDFLD(KLJUC) RESP(STATCICS);
 IF STATCICS=DFHRESP(NOTFND) THEN
   RC=SENDP('ERRCICS:UNKNOWN PROGID ' || TCPBUF.PROGID,25);
 ELSE IF STATCICS¬=DFHRESP(NORMAL) THEN DO;
   STATCICSP=STATCICS;
   MESSAGE1='ERRCICS:FILE (IØØ2VSAM) ERROR DFHRESP=' || STATCICSP;
   RC=SENDP(MESSAGE1,LENGTH(MESSAGE1));
 END;
 ELSE DO;
   IF TCPBUFLEN¬=(PRGINPUL+2) THEN
     RC=SENDP('ERRCICS:INVALID SIZE OF PARAMETER CICS INPUT',44);
   ELSE IF PRGINPUL>5ØØ THEN
     RC=SENDP('ERRCICS:INVALID VALUE OF PRGINPUL IN FILE',41);
   ELSE DO;
     COMM=TCPBUF.DATA;
     EXEC CICS LINK PROGRAM(PRGNAME) COMMAREA(COMM) RESP(STATCICS);
     IF STATCICS¬=DFHRESP(NORMAL) THEN DO;
        IF STATCICS=DFHRESP(PGMIDERR) THEN
          MESSAGE1='LINK PROGRAM ERROR DFHRESP=PGMIDERR';
```

```
        ELSE DO;
          STATCICSP=STATCICS;
          MESSAGE1='LINK PROGRAM ERROR DFHRESP=' || STATCICSP;
        END;
        MESSAGE1='ERRCICS:' || MESSAGE1;
        RC=SENDP(MESSAGE1,LENGTH(MESSAGE1));
      END;
      ELSE DO; /* OK */
        RC=SENDP(COMM,PRGOUTPL);
        IF RC=Ø THEN RC=SENDP('END',3);
      END;
    END;
  END;
  RETURN(RC);
 END EXELINK;

 END TCPIPØ1;
```

MODULE1.BAS

```
Public Buffer As String
Public TCPRes As Integer
Public Tran As String
Public SendMess As String
Public Busy As Boolean
```

TCPCLASS1.CLS

TCPCLASS1.CLS is the class module.

```
Public Function RunCICS(PID As String, CICSInput As String, CICSOutput
As String, ErrDesc As String) As Integer
    Tran = "TCP1"
    SendMess = PID & CICSInput
    fTCPIP.Show 1
    If TCPRes = Ø Then
        CICSOutput = Buffer
        ErrDesc = ""
    Else
        ErrDesc = Buffer
        RunCICS = TCPRes
    End If
End Function
```

FTCPIP.FRM

FTCPIP.FRM is the form with WINSOCK and TIMER components.

```
Option Explicit
Dim SockMsg As String
Dim SockStart As String
Dim SockEnd As String
Dim SockIBM As String
Dim ConnStart As Boolean
Dim ConnIBM As Boolean

Private Sub Finish()
    If Winsock1.State <> Ø Then Winsock1.Close
    Unload Me
End Sub

Private Sub Form_Activate()
If Not Busy Then
    Busy = True
    SockMsg = ""
    Buffer = ""
    ConnStart = False
    ConnIBM = False
    TCPRes = Ø
    Winsock1.Connect
' DEFINE TIMEOUT 1Øs
    Timer1.Interval = 1ØØØØ
Else
    TCPRes = -6
    Buffer = "BUSY"
    Unload Me
End If
End Sub

Private Sub Form_Load()
    Me.Caption = "CICS PROGRAM IS RUNNING. PLEASE WAIT ..."
    SockStart = "#B~B#"
    SockEnd = "#E~E#"
    SockIBM = "#S~S#"
    Winsock1.Protocol = sckTCPProtocol
    Winsock1.RemotePort = 3ØØ2
    Winsock1.RemoteHost = "4.3.2.1"
    Me.Height = Ø
    Me.Top = Ø
    Me.Left = Ø
End Sub
Private Sub Form_Unload(Cancel As Integer)
    Busy = False
End Sub

Private Sub Timer1_Timer()
    Buffer = "TIMEOUT"
    TCPRes = -1
    Finish
```

```
End Sub

Private Sub Winsock1_Connect()
    Winsock1.SendData Tran
End Sub

Private Sub Winsock1_DataArrival(ByVal bytesTotal As Long)
Dim s1 As String
Dim Res As Integer
Dim TCPData As String
    Winsock1.GetData s1
    SockMsg = SockMsg & s1
    If Not ConnIBM Then
        Res = InStr(1, SockMsg, SockIBM)
        If Res > Ø Then
            SockMsg = Mid(SockMsg, Res + Len(SockIBM))
            s1 = SendMess
            Winsock1.SendData s1
            ConnIBM = True
        End If
    Else ' Connected to IBM
        While InStr(1, SockMsg, SockEnd) > Ø
            Res = InStr(1, SockMsg, SockStart)
            If Res > Ø Then
                If ConnStart Then
                    TCPRes = -4
                    Buffer = "STARTx2"
                    Finish
                Else
                    ConnStart = True
                    SockMsg = Mid(SockMsg, Res + Len(SockStart))
                End If
            End If
            Res = InStr(1, SockMsg, SockEnd)
            If Res > Ø Then
                If ConnStart Then
                    ConnStart = False
                    TCPData = Left(SockMsg, Res - 1)
                    SockMsg = Mid(SockMsg, Res + Len(SockEnd))
                    If TCPData = "END" Then
                        Finish
                    ElseIf Left(TCPData, 3) = "ERR" Then
                        TCPRes = -3
                        Buffer = Mid(TCPData, 4)
                        Finish
                    Else
                        Buffer = Buffer & TCPData
                    End If
                Else
                    TCPRes = -5
```

```
                    Buffer = "END WITHOUT START"
                    Finish
                End If
            End If
        Wend
    End If ' CONNIBM
End Sub

Private Sub Winsock1_Error(ByVal Number As Integer, Description As
String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As
String, ByVal HelpContext As Long, CancelDisplay As Boolean)
    TCPRes = -2
    Buffer = Hex(Number) & Description
    Finish
End Sub
```

In practice, one socket may pass two (or more) times through the DataArrival of Winsock1 procedure. This is more evident with large sockets. This means that it's possible to divide one socket into two (or more) smaller sockets (pieces). Concatenating the pieces presents a problem, however, because there's no way of knowing when we've received a complete response. One solution is to insert start and stop delimiters inside a socket before sending them from CICS. If these can be detected on the PC, we can be sure the socket is all there.

An alternative solution is to check the size of the socket after joining the pieces. However, this doesn't work well because different CICS programs have differently-sized common areas. With this method, if we want to append a new CICS program, we would have to append commands for checking its size in our TCPActiveX.dll and install it on all the PCs!

Note that in CICS transaction TCP1, I had to use the DELAY(100) command. Without this command, we sometimes get unhandled errors in TCPActiveX. I think this is caused by the PC getting a 'close socket' message before receiving all the data.

Note also that TCPActiveX.dll has a timer which generates a timeout after 10 seconds and returns control to the program that called the RunCICS function.

After receiving the PID and CICSInput parameters, the TCP1 transaction searches a VSAM file (I002VSAM) for the

corresponding CICS program name (PRGNAME), the length of input data (PRGINPUL), and the length of common area (PRGOUTPL (>=PRGINPUL)), and checks them. Instead of this VSAM file you could of course use a memory array with the same structure. If you do that, however, you would have to change and recompile TCP1 when you wanted to append a new CICS program.

Dejan Jelic
Programmer, Postal Savings Bank (Yugoslavia)

The FTPPUT utility

Since IBM added the TCP/IP protocol and the traditional TCP/IP tools to the mainframe environment, the popular tools of the Unix world have become available to the mainframe user. These tools offer the programmer a great deal of help with common activities. ftp (File Transfer Protocol) is one of the more recognizable tools in the TCP/IP tool suite, and has tremendous potential to improve the quality of life for many mainframe users.

ftp is similar to many other tools for moving files between hosts. You may be familiar with the traditional mainframe tools for moving MVS files (sequential files and PDS members) to non-mainframe platforms. These are IND$FILE, the older (and less frequently seen) IRMA FTTSO, the newer (and now stabilized) ISPF Workstation, and the proprietary Connect: Direct (NDM). Each of these has pros and cons, and, in my opinion, most have drawbacks and lack flexibility and interoperability.

The OS/390 implementation of ftp allows a mainframe to participate with any host that supports ftp (almost every platform these days). Even though the ftp command syntax is very similar on most platforms (it's governed by RFCs), I still found the process could use a little help on the OS/390 platform – not to extend or change the ftp implementation, but to integrate it with existing processes.

After trying to use ftp in existing production processes (batch JCL), I found a number of problems:

- It was difficult to implement without significant tailoring.
- ftp jobs were susceptible to frequent modification (especially those using relative GDGs).
- Condition code processing was not intuitively obvious.

I also found it less intuitive to teach to interactive TSO users with a purely mainframe background. Like many TSO/ISPF users, I commonly wanted to bring things to the mainframe for editing. Even after five years of doing Unix Sysadmin work, I still refuse to use the Unix vi editor. While working on multi-platform projects I frequently wanted to get files from or put files on other platforms while editing a mainframe dataset. Finally, under all these circumstances, I wanted all ftp processes to be initiated from the mainframe.

This resulted in the creation of three REXX EXECs that I and others have found useful, presented here as the FTPPUT batch tool (see Figure 1).

FTPPUT

The FTPPUT EXEC is a batch implementation of ftp that allows the user to identify one or multiple sequential datasets, relative or absolute GDGs, VSAM datasets, HFS files, and PDSs in a single step of JCL. The program uses EXEC card parms to specify the destination (IP address and directory) and 'remote' userid/password for all the files in the job and FTP* DD statements to identify all the files to be transferred. This implementation facilitates incorporation in PROCs by allowing symbolics to be placed in the EXEC card parms and traditional DD statements for all datasets to be transferred as well as the use of dependable condition code processing.

FTPPUT depends on an ftp server running on the target platform. All Unix platforms run an ftp server. Several are available for the Microsoft operating systems (Win9*x*, NT, 2000,

EXEC	Description
FTPPUT	Batch implementation of FTP PUT processing
@FTPPUT	ISPF Edit Macro to FTP PUT the contents of an edit session
@FTPGET	ISPF Edit Macro to FTP GET a foreign file into an edit session

Figure 1: FTPPUT batch tool – REXX EXECs

XP, etc). A-FTP and WSFTP are two examples of ftp servers that can be downloaded from sources like Download.com to make this a tool to move files to and from Microsoft workstations. This was actually the original inspiration for this utility – I wanted to download one or more PDSs and sequential files to my workstation and didn't want to 'hang' my TSO session or rewrite the ftp subcommand input for a job every time.

If FTPPUT is used to send files to a workstation, the DNS name or IP address must be known. In many shops, not all workstations have a name in the DNS, and if DHCP is used the IP address can change. NT and OS/2 users can use the IPCONFIG command (from a Command Prompt) and Win9*x* users can use the WINIPCFG command (from Start/Run) to get the workstation's IP address.

FTPPUT will format and execute an ftp PUT subcommand for sequential files and create a file name on the target platform identical to the MVS dataset name. Relative Gags will create a file name on the target platform the same as the absolute GDG name. If the dataset is a PDS, FTPPUT will format and execute an ftp MPUT * subcommand which will copy all of the members of the PDS to individual files on the target platform in the target directory. In this case, the file names will be the same as the member names. All member file names on the target platform will default to upper case. I tried generating a single ftp PUT for every member to provide upper/lower case flexibility, but performance was worse and the visual recognition of files that came from the mainframe became a downstream benefit.

If the dataset is VSAM, SORT is invoked to create a sequential VB file to ftp PUT with the same name as the original VSAM dataset. No special processing of VSAM is done; if it contains packed or binary data, it will be fairly unusable downstream. On the other hand, if it's primarily character data and zoned decimal data, it is readable downstream. If the file is an HFS file, the 'basename' will be FTP PUT and the case will be retained. If the file is an HFS directory, the entire contents of the HFS directory will be FTP MPUT and the case will be retained.

All transfers default to ASCII. ASCII transfers using ftp are identical to IND$FILE TEXT transfers. One of the nice things about ftp is that ASCII to EBCDIC conversion takes place automagically. I've also found that the TCP/IP translation tables used by ftp seem more accurate than IND$FILE ('|' is no longer converted to ']'). Since my original inspiration involved moving REXX source code PDSs around, this was an unexpected but welcome benefit.

The JCL to run FTPPUT requires a few things:

- An EXEC card for IKJEFT01 with parms for the FTPPUT REXX EXEC, target IP address, target directory, userid, and password.
- Standard DD statements used in batch TSO (SYSEXEC, SYSTSPRT, and SYSTSIN).
- DD statements beginning with FTP* to identify all the datasets to be sent.

There are some limitations, namely:

- The FTP* DD statements do not support concatenation.
- Direct access datasets are not supported.
- Since unsupported datasets will be bypassed with only an RC=4, it may be advisable to limit one dataset per FTPPUT step if you must guarantee that a dataset was successfully sent.
- Since FTP PUT can send only catalogued datasets, there cannot be &&TEMP datasets on the FTP* DDs.

- Since Unix is case sensitive, you may want to turn ‘CAPS OFF’ on your JCL member, so the target directory can be located on the target host.
- FTPPUT generates ftp MPUT statements for PDSs and will always copy member names to upper case file names on the target system. The CASE parm will not change this situation.
- All HFS files will be transferred with the SAME case. The CASE parm will not change this situation.

SAMPLE JCL TO RUN FTPPUT

The sample JCL to run FTPPUT is as follows:

```
//FTPPUT   EXEC PGM=IKJEFTØ1,
//         PARM='FTPPUT 1Ø.1.1.1 /data/stuff'
//SYSEXEC  DD   DSN=exec.pds,DISP=SHR
//SYSTSPRT DD   SYSOUT=*
//OUTPUT   DD   SYSOUT=*
//SYSTSIN  DD   DUMMY
//FTPPDS   DD   DSN=pds.to.send,DISP=SHR
//FTPSEQ1  DD   DSN=seq.file1.to.send,DISP=SHR
//FTPSEQ2  DD   DSN=seq.file2.to.send,DISP=SHR
//FTPGDG1  DD   DSN=rel.gdg1.to.send(Ø),DISP=SHR
//FTPVSAM  DD   DSN=vsam.cluster,DISP=SHR
//FTPHFILE DD   PATH='/u/mystuff/configs/parm.cfg'
//FTPHDIR  DD   PATH='/u/mystuff/scripts'
```

Parm	Status	Description
IP_Addr	Required	IP address or DNS name of target host
Directory	Required	Fully qualified directory of target directory on target host
Userid	Optional	Userid, defaults to "anonymous"
Password	Optional	Password, defaults to the current userid
Mode	Optional	FTP Mode (ASCII or BINARY), defaults to ASCII
Case	Optional	Target filename case (UPPER or LOWER), defaults to UPPER and does not apply for PDS or HFS transfers

Figure 2: FTPPUT parms

DDNAME	Status	Description
SYSEXEC	Required	Location of the FTPPUT REXX EXEC
SYSTSPRT	Required	REXX and TSO output messages
SYSTSIN	Required	REXX and TSO input
OUTPUT	Optional	FTP messages, dynamically allocated if not in the JCL
FTP*	Required	At least one FTP* DD statement is required, last 5 characters of the DDNAME are selected by the user
DIAGMSGS	Required	All diagnostic messages from FTPPUT

Figure 3: FTPPUT DD statements

Dataset type	FTP action	Notes
Sequential datasets	FTP PUT	Same name on target
Absolute GDG	FTP PUT	Same name on target
Relative GDG	FTP PUT	Absolute GDG name on target
PDS	FTP MPUT	Upper case membernames on target
VSAM	FTP PUT	Same name on target
HFS file	FTP PUT	Same name, same case on target
HFS directory	FTP MPUT	Case retained on target
All others	Bypass	RC=4 and continues to the next file

Figure 4: FTPPUT actions

Return code	Description
4	Unsupported dataset type
12	TSO Command failure, see SYSTSPRT for messages
Over 12	See IP User's Guide under FTP EXIT Return Codes

Figure 5: FTPPUT return codes

FTPPUT PARMS, STATEMENTS, ACTIONS, AND RETURN CODES

Figures 2, 3, 4, and 5 show the FTPPUT parms, statements, actions, and return codes respectively.

FTPPUT

```
/*********************************************************************/
/*                              REXX                                 */
/*********************************************************************/
/* Purpose: To FTP PUT all datasets found allocated to FTP* DD's     */
/*-------------------------------------------------------------------*/
/* Syntax:  FTPPUT ipaddr dir                                        */
/*-------------------------------------------------------------------*/
/* Parms: ipaddr     - IP Address of destination (required)          */
/*        dir        - Target directory (defaults to /)              */
/*        user       - UserId on destination (defaults to anonymous) */
/*        pass       - Password for Userid (defaults to userid())    */
/*        mode       - ASCII or BINARY (defaults to ASCII)           */
/*        case       - UPPER or LOWER (defaults to UPPER)            */
/*                     PDS members are always in UPPER case          */
/*                                                                   */
/* Notes: Intended to run in batch                                   */
/*                                                                   */
/* Include as many FTPxxxxx DD statements as required.  All will be  */
/* FTP'd to the same IP Address.  Does NOT support concatenation.    */
/*                                                                   */
/* All DD's beginning with 'FTP' will be sent.                       */
/*                                                                   */
/* If DSN is a sequential file, then a FTP PUT will be executed      */
/* If DSN is a PDS, then a FTP MPUT * will be executed               */
/* If file is an HFS file, then a FTP PUT will be executed           */
/* If file is an HFS directory, then FTP MPUT * will be executed     */
/*                                                                   */
/* Remember to set "CAPS OFF" in your JCL/PROC member if lower case  */
/* is required (i.e UNIX is case sensitive)                          */
/*                                                                   */
/* //FTPPUT   EXEC PGM=IKJEFT01,PARM='FTPPUT 10.1.1.1 c:\data\stuff' */
/* //SYSEXEC  DD   DSN=exec.pds,DISP=SHR                             */
/* //SYSTSPRT DD   SYSOUT=*                                          */
/* //OUTPUT   DD   SYSOUT=*  (optional, FTP will create 1 per file)  */
/* //DIAGMSGS DD   SYSOUT=*                                          */
/* //SYSTSIN  DD   DUMMY                                             */
/* //FTPPDS   DD   DSN=pds.to.send,DISP=SHR                          */
/* //FTPSEQ1  DD   DSN=seq.file1.to.send,DISP=SHR                    */
/* //FTPSEQ2  DD   DSN=seq.file2.to.send,DISP=SHR                    */
/* //FTPGDG1  DD   DSN=rel.gdg1.to.send(0),DISP=SHR                  */
/* //FTPVSAM  DD   DSN=vsan.cluster.dsn,DISP=SHR                     */
/* //FTPHFSF  DD   PATH='/u/mystuff/hello.sh'                        */
/* //FTPHFSD  DD   PATH='/u/mystuff/sql'                             */
/*                                                                   */
/* Receiving system MUST be running an FTP Server.  PC users can     */
/* download the public domain A-FTP Server from Download.com         */
/*                                                                   */
/* PC users can use the IPCONFIG command on NT and WINIPCFG on Win9x */
/* to get the IP address for the workstation.                        */
```

```
/*                                                                   */
/* All files are sent as ASCII.  Filenames on the receiving system   */
/* will be the same as the DSN (same as the member name if a PDS).   */
/* GDG's will use absolute GDG name.                                 */
/*                                                                   */
/* PS, PO, VSAM , HFS files and HFS directories are supported.  All  */
/* others will be bypassed.  A non-zero RC will result if a dataset  */
/* is bypassed.                                                      */
/*                                                                   */
/* Don't send multiple datasets if you need single dataset level     */
/* confirmation in your JCL condition codes.                         */
/*                                                                   */
/*********************************************************************/
/*                      Change Log                                   */
/*                                                                   */
/*  Author       Date      Reason                                    */
/*  --------    --------- ----------------------------------------- */
/*  R. Zenuk    Apr 2000   Initial Creation                          */
/*  R. Zenuk    09/24/01   Combine FTPMPUT and FTPGPUT               */
/*  R. Zenuk    09/27/01   1) Deal with non-PS and PO datasets       */
/*                         2) Lower case defaults and FTP subcmds    */
/*                            for cleaner UNIX support               */
/*                         3) Improve shutdown and cleanup logic     */
/*  R. Zenuk    10/11/01   Incorporated RCEXIT                       */
/*  R. Zenuk    10/17/01   Fixed MAXRC problem                       */
/*  R. Zenuk    10/30/01   Added updated RCEXIT and STDENTRY code    */
/*  R. Zenuk    11/29/01   Added updated RCEXIT and STDENTRY code    */
/*  R. Zenuk    06/20/02   Added MODE and CASE parms and newmodel    */
/*  R. Zenuk    07/18/02   Incorporated @REFRESH support             */
/*  R. Zenuk    09/07/02   Added HFS support                         */
/*  R. Zenuk    09/11/02   Added VSAM support                        */
/*                                                                   */
/*********** @REFRESH BEGIN START    2002/09/11 00:41:39 ************/
/* Standard startup activities                                       */
/*********************************************************************/
 call time 'r'
 parse arg parms
 signal on syntax name trap
 signal on failure name trap
 signal on novalue name trap
 probe = 'NONE'
 modtrace = 'NO'
 modspace = ''
 call stdentry 'DIAGMSGS'
 module = 'MAINLINE'
 push trace() time('L') module 'From:' sigl 'Parms:' sparms
 if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
 call modtrace 'START' sigl
/*********** @REFRESH END   START    2002/09/11 00:41:39 ************/
/* Accept all the parms                                              */
```

```
/*********************************************************************/
 parse arg ipaddr ftpdir ftpuser ftppass ftpmode ftpcase
 if ipaddr  = '' then call rcexit 999 'Required IP Address is missing'
 if ftpdir  = '' then ftpdir  = '/'
 if ftpuser = '' then ftpuser = 'anonymous'
 if ftppass = '' then ftppass = userid()
 if ftpmode = '' then ftpmode = 'ASCII'
 if ftpcase = '' then ftpcase = 'UPPER'
/*********************************************************************/
/* Loop through all DD's looking for FTP DD's                        */
/*********************************************************************/
 ddnum = ddlist()
 fddlist = ddlist
 ftpdds = 0
 do loop=1 to ddnum
    ftpdd = word(fddlist,loop)
    if abbrev(ftpdd,'FTP',3) then
       do
        ftpdds = ftpdds + 1
/*********************************************************************/
/* Get filename in case this is HFS                                  */
/*********************************************************************/
        ftpfile = 'garbage'
        fcount = dddsns(ftpdd)
        ftpfile = dddsns
        stat.ST_TYPE = 0
/*********************************************************************/
/* Get file attributes for a USS FTP DD's                            */
/*********************************************************************/
        if substr(ftpfile,1,1) = '/' then
           do
            call rcexit syscalls('ON') 'USS Initialization error'
            URC = usswrap("stat" ftpfile "stat.")
            sysdsorg = 'HFS'
           end
        else
/*********************************************************************/
/* Get DSN attributes for all MVS FTP DD's                           */
/*********************************************************************/
           do
            FILERC = listdsi(ftpdd "FILE")
            if FILERC <> 0 & sysreason <> 12 then MAXRC = FILERC
           end
/*********************************************************************/
/* Is this a PDS?                                                    */
/*********************************************************************/
        select
          when sysdsorg = 'PO' then
               do
                input.4 = "lcd '"sysdsname"'"
```

```
               input.5 = "mput *"
               input.6 = "quit"
               putcmd = input.5
              end
/*********************************************************************/
/* Is this a sequential file?                                        */
/*********************************************************************/
          when sysdsorg = 'PS' then
               do
                if ftpcase = 'LOWER' then
                   do
                    lower = xrange('a','z')
                    upper = xrange('A','Z')
                    targetfile = translate(sysdsname,lower,upper)
                   end
                else
                   do
                    targetfile = sysdsname
                   end
                input.4 = "put" "'"sysdsname"'" targetfile
                input.5 = "quit"
                putcmd = input.4
               end
/*********************************************************************/
/* Is this a VSAM KSDS?                                              */
/*********************************************************************/
          when sysdsorg = 'VS' then
               do
                vsamdsn = sysdsname
/*********************************************************************/
/* Allocate a temp SEQ DSN by prefixing VSAMDSN with current userid  */
/*********************************************************************/
               tempdsn = userid()'.'vsamdsn
               call tsotrap "ALLOC F(SORTIN) DA('"sysdsname"') SHR REU"
               call tsotrap "ALLOC F(SORTOUT) DA('"tempdsn"')",
                            "NEW UNIT(SYSDA) SPACE(5 1Ø) CYLINDERS"
/*********************************************************************/
/* Invoke a SORT COPY to copy the records to the temp SEQ DSN        */
/*********************************************************************/
               sysin.1 = ' SORT FIELDS=COPY'
               call viodd "SYSIN"
               call tsotrap "EXECIO * DISKW SYSIN (STEM SYSIN. FINIS"
               parms = ''
               address ATTCHMVS "SORT" "PARMS"
/*********************************************************************/
/* PUT the temporary SEQ DSN as the VSAMDSN                          */
/*********************************************************************/
               if ftpcase = 'LOWER' then
                  do
                   lower = xrange('a','z')
```

```
                upper = xrange('A','Z')
                vsamdsn = translate(vsamdsn,lower,upper)
               end
             input.4 = "put '"tempdsn"'" vsamdsn
             input.5 = "quit"
             putcmd = input.4
             call tsotrap "FREE F(SORTIN)"
             call tsotrap "FREE F(SYSIN)"
            end
/*********************************************************************/
/* Is this an HFS file?                                              */
/*********************************************************************/
         when substr(ftpfile,1,1) = '/' & stat.ST_TYPE = 3 then
              do
               slash = lastpos('/',ftpfile)
               path = substr(ftpfile,1,slash)
               basename = substr(ftpfile,slash+1,length(ftpfile)-slash)
               sysdsorg = 'HF'
               sysdsname = ftpfile
               input.4 = "lcd" path
               input.5 = "put" basename basename
               input.6 = "quit"
               putcmd = input.5
              end
/*********************************************************************/
/* Is this an HFS directory?                                         */
/*********************************************************************/
         when substr(ftpfile,1,1) = '/' & stat.ST_TYPE = 1 then
              do
               sysdsorg = 'HD'
               sysdsname = ftpfile
               input.4 = "lcd" ftpfile
               input.5 = "mput *"
               input.6 = "quit"
               putcmd = input.5
              end
/*********************************************************************/
/* Otherwise this is an unsupported file type                        */
/*********************************************************************/
         otherwise
              do
               say left(ftpdd,8) sysdsorg 'RC=4' sysdsname sysmsglvl2
               MAXRC = 4
               iterate loop
              end
        end
/*********************************************************************/
/* Terminate the USS environment                                     */
/*********************************************************************/
        call rcexit syscalls('OFF') 'USS Termination error'
```

```
      end
   else
      do
       iterate loop
      end
/*********************************************************************/
/* Write FTP subcommands to INPUT file                               */
/*********************************************************************/
   input.1 = ftpuser ftppass
   input.2 = ftpmode
   input.3 = "cd" ftpdir
   call viodd 'INPUT'
/*********************************************************************/
/* Unload the parentage stack to avoid FTP problems                  */
/*********************************************************************/
   do deq=1 to queued()
      pull stackinfo
      tempq.deq = stackinfo
   end
/*********************************************************************/
/* Invoke FTP with '-i' to avoid prompting with MPUT                 */
/*********************************************************************/
   address TSO "FTP -i" ipaddr "(EXIT"
   EXITRC = RC
   say left(ftpdd,8) sysdsorg 'RC='EXITRC sysdsname putcmd
/*********************************************************************/
/* Cleanup                                                           */
/*********************************************************************/
   drop input.
/*********************************************************************/
/* VSAM cleanup                                                      */
/*********************************************************************/
   if sysdsorg = 'VS' then call tsotrap "FREE F(SORTOUT) DELETE"
/*********************************************************************/
/* Reload the parentage stack                                        */
/*********************************************************************/
   do req=deq-1 to 1 by -1
      push tempq.req
   end
/*********************************************************************/
/* If EXITRC is non-zero, then write the INPUT to DIAGMSGS           */
/*********************************************************************/
   if EXITRC <> Ø then
      do
       call tsotrap "EXECIO * DISKR INPUT (STEM ERRINPUT. FINIS"
       call saydd msgdd 1 'INPUT for' sysdsname putcmd 'RC='EXITRC
       do err=1 to errinput.Ø
          call saydd msgdd Ø errinput.err
       end
      end
```

```
 end
/*********************************************************************/
/* Shutdown                                                          */
/*********************************************************************/
 shutdown: nop
/*********************************************************************/
/* Cleanup the INPUT dataset                                         */
/*********************************************************************/
           call tsotrap "FREE F(INPUT)"
/*********************************************************************/
/* Summary                                                           */
/*********************************************************************/
           say
           if ftpdds > 1 then
              say ftpdds 'datasets processed'
           else
              say ftpdds 'dataset processed'
           say
/*********** @REFRESH BEGIN STOP     2002/08/03 08:42:33 *************/
/* Shutdown message and terminate                                    */
/*********************************************************************/
           call stdexit time('e')
/*********** @REFRESH END   STOP     2002/08/03 08:42:33 *************/
/*********** @REFRESH BEGIN SUBBOX   2002/08/15 12:46:24 *************/
/*                                                                   */
/* Internal Subroutines provided in FTPPUT                           */
/*                                                                   */
/* RCEXIT   - Exit on non-zero return codes                          */
/* TRAP     - Issue a common trap error message using rcexit         */
/* ERRMSG   - Build common error message with failing line number    */
/* STDENTRY - Standard Entry logic                                   */
/* STDEXIT  - Standard Exit logic                                    */
/* MSG      - Determine whether to SAY or ISPEXEC SETMSG the message */
/* DDCHECK  - Determine if a required DD is allocated                */
/* DDDSNS   - Returns number of DSNs in a DD and populates DDDSNS    */
/* DDLIST   - Returns number of DD's and populates DDLIST variable   */
/* TSOTRAP  - Capture the output from a TSO command in a stem        */
/* SAYDD    - Print messages to the requested DD                     */
/* JOBINFO  - Get job related data from control blocks               */
/* PTR      - Pointer to a storage location                          */
/* STG      - Return the data from a storage location                */
/* CMDTIMER - Time a command, if this is a background run            */
/* VIODD    - EXECIO a stem into a TEMP PDS                          */
/* USSWRAP  - Wrapper for USS API commands                           */
/* MODTRACE - Module Trace                                           */
/*                                                                   */
/*********** @REFRESH END   SUBBOX   2002/08/15 12:46:24 *************/
/*********** @REFRESH BEGIN RCEXIT   2002/08/15 15:28:39 *************/
/* RCEXIT   - Exit on non-zero return codes                          */
/*-------------------------------------------------------------------*/
```

```
/* EXITRC   - Return code to exit with (if non zero)                */
/* ZEDLMSG  - Message text for it with for non zero EXITRC's        */
/*******************************************************************/
 rcexit: parse arg EXITRC zedlmsg
         if EXITRC <> Ø then
            do
             trace 'o'
/*******************************************************************/
/* If execution environment is ISPF then VPUT ZISPFRC              */
/*******************************************************************/
             if execenv = 'TSO' | execenv = 'ISPF' then
                do
                 if ispfenv = 'YES' then
                    do
                     zispfrc = EXITRC
/*******************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules  */
/*******************************************************************/
                     address ISPEXEC "VPUT (ZISPFRC)"
                    end
                end
/*******************************************************************/
/* If a message is provided, wrap it in date, time and EXITRC      */
/*******************************************************************/
             if zedlmsg <> '' then
                do
                 zedlmsg = time('L') execname zedlmsg 'RC='EXITRC
                 call msg zedlmsg
                end
/*******************************************************************/
/* Write the contents of the Parentage Stack                       */
/*******************************************************************/
             stacktitle = 'Parentage Stack Trace ('queued()' entries):'
/*******************************************************************/
/* Write to MSGDD if background                                    */
/*******************************************************************/
             if tsoenv = 'BACK' then
                do
                 call saydd msgdd 1 zedlmsg
                 call saydd msgdd 1 stacktitle
                end
             else
/*******************************************************************/
/* Write to the ISPF Log if foreground                             */
/*******************************************************************/
                do
                 zerrlm = zedlmsg
                 address ISPEXEC "LOG MSG(ISRZØØ3)"
                 zerrlm = center(' 'stacktitle' ',78,'-')
                 address ISPEXEC "LOG MSG(ISRZØØ3)"
```

```
               end
/*********************************************************************/
/* Unload the Parentage Stack                                        */
/*********************************************************************/
            do queued()
               pull stackinfo
               if tsoenv = 'BACK' then
                  call saydd msgdd Ø stackinfo
               else
                  do
                   zerrlm = stackinfo
                   address ISPEXEC "LOG MSG(ISRZØØ3)"
                  end
            end
/*********************************************************************/
/* Put a terminator in the ISPF Log for the Parentage Stack          */
/*********************************************************************/
            if tsoenv = 'FORE' then
               do
                zerrlm = center(' 'stacktitle' ',78,'-')
                address ISPEXEC "LOG MSG(ISRZØØ3)"
               end
/*********************************************************************/
/* Signal SHUTDOWN.  SHUTDOWN label MUST exist in the program        */
/*********************************************************************/
             signal shutdown
            end
         else
            return
/*********** @REFRESH END   RCEXIT   2ØØ2/Ø8/15 15:28:39 *************/
/*********** @REFRESH BEGIN TRAP     2ØØ2/Ø8/Ø7 11:48:14 *************/
/* TRAP     - Issue a common trap error message using rcexit         */
/*-------------------------------------------------------------------*/
/* PARM     - N/A                                                    */
/*********************************************************************/
 trap: traptype = condition('C')
       if traptype = 'SYNTAX' then
          msg = errortext(RC)
       else
          msg = condition('D')
       trapline = strip(sourceline(sigl))
       msg = traptype 'TRAP:' msg', Line:' sigl '"'trapline'"'
       call rcexit 666 msg
/*********** @REFRESH END   TRAP     2ØØ2/Ø8/Ø7 11:48:14 *************/
/*********** @REFRESH BEGIN ERRMSG   2ØØ2/Ø8/1Ø 16:53:Ø4 *************/
/* ERRMSG   - Build common error message with failing line number    */
/*-------------------------------------------------------------------*/
/* ERRLINE  - The failing line number passed by caller from SIGL     */
/* TEXT     - Error message text passed by caller                    */
/*********************************************************************/
```

```
 errmsg: nop
         parse arg errline text
         return 'Error on statement' errline',' text
/*********** @REFRESH END   ERRMSG   2002/08/10 16:53:04 *************/
/*********** @REFRESH BEGIN STDENTRY 2002/09/11 01:48:55 *************/
/* STDENTRY - Standard Entry logic                                   */
/*-------------------------------------------------------------------*/
/* MSGDD    - Optional MSGDD used only in background                 */
/*********************************************************************/
 stdentry: module = 'STDENTRY'
           if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
           parse arg sparms
           push trace() time('L') module 'From:' sigl 'Parms:' sparms
           arg msgdd
           parse upper source . . execname . execdsn . . execenv .
/*********************************************************************/
/* Startup values                                                    */
/*********************************************************************/
           EXITRC = 0
           MAXRC = 0
           ispfenv = 'NO'
           popup = 'NO'
           lockpop = 'NO'
           keepstack = 'NO'
/*********************************************************************/
/* Determine environment                                             */
/*********************************************************************/
           if substr(execenv,1,3) <> 'TSO' & execenv <> 'ISPF' then
              tsoenv = 'NONE'
           else
              do
               tsoenv = sysvar('SYSENV')
              "ISPQRY"
               if RC = 0 then ispfenv = 'YES'
              end
/*********************************************************************/
/* MODTRACE must occur after the setting of ISPFENV                  */
/*********************************************************************/
           call modtrace 'START' sigl
/*********************************************************************/
/* Startup message                                                   */
/*********************************************************************/
           lpar = mvsvar('SYSNAME')
           startmsg = execname 'started' date() time() 'on' lpar
           if tsoenv = 'BACK' then
              do
               jobname = mvsvar('SYMDEF','JOBNAME')
               jobinfo = jobinfo()
               parse var jobinfo jobtype jobnum .
               say jobname center(' 'startmsg' ',61,'-') jobtype jobnum
```

```
              say
/*********************************************************************/
/* If MSGDD is provided, write the STARTMSG and SYSEXEC DSN to MSGDD */
/*********************************************************************/
              if msgdd <> '' then
                 do
                  call saydd msgdd 1 startmsg
                  x = listdsi('SYSEXEC' 'FILE')
                  call saydd msgdd Ø execname 'loaded from' sysdsname
/*********************************************************************/
/* If there are PARMS, write them to the MSGDD                       */
/*********************************************************************/
                  if parms <> '' then
                     call saydd msgdd Ø 'Parms:' parms
/*********************************************************************/
/* If there is a STEPLIB, write the STEPLIB DSN MSGDD                */
/*********************************************************************/
                  if listdsi('STEPLIB' 'FILE') = Ø then
                     do
                      steplibs = dddsns('STEPLIB')
                      call saydd msgdd Ø 'STEPLIB executables loaded',
                          'from' word(dddsns,1)
                      if dddsns('STEPLIB') > 1 then
                         do
                          do stl=2 to steplibs
                             call saydd msgdd Ø copies(' ',31),
                                  word(dddsns,stl)
                          end
                         end
                     end
                 end
              end
           pull tracelvl . module . sigl . sparms
           call modtrace 'STOP' sigl
           interpret 'trace' tracelvl
           return
/********** @REFRESH END   STDENTRY 2ØØ2/Ø9/11 Ø1:48:55 ************/
/********** @REFRESH BEGIN STDEXIT  2ØØ2/Ø9/11 Ø1:ØØ:51 ************/
/* STDEXIT  - Standard Exit logic                                    */
/*-------------------------------------------------------------------*/
/* ENDTIME  - Elapsed time                                           */
/* Note: Caller must set KEEPSTACK if the stack is valid             */
/*********************************************************************/
 stdexit: module = 'STDEXIT'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg endtime
          endmsg = execname 'ended' date() time() format(endtime,,1)
```

```
/*********************************************************************/
/* if MAXRC is greater then EXITRC then set EXITRC to MAXRC          */
/*********************************************************************/
          if MAXRC > EXITRC then EXITRC = MAXRC
          endmsg = endmsg 'on' lpar 'RC='EXITRC
          if tsoenv = 'BACK' then
             do
              say
              say jobname center(' 'endmsg' ',61,'-') jobtype jobnum
              if msgdd <> '' then
                 do
                  call saydd msgdd 1 execname 'ran in' endtime 'seconds'
                  call saydd msgdd Ø endmsg
                 end
             end
/*********************************************************************/
/* Remove STDEXIT and MAINLINE Parentage Stack entries, if there     */
/*********************************************************************/
          if queued() > Ø then pull . . module . sigl . sparms
          if queued() > Ø then pull . . module . sigl . sparms
          call modtrace 'STOP' sigl
/*********************************************************************/
/* if the Parentage Stack is not empty, display its contents         */
/*********************************************************************/
          if queued() > Ø & keepstack = 'NO' then
             do
              say 'Leftover Parentage Stack Entries:'
              say
              do queued()
                 pull stackundo
                 say stackundo
              end
              EXITRC = 1
             end
/*********************************************************************/
/* Exit                                                              */
/*********************************************************************/
          exit(EXITRC)
/********** @REFRESH END   STDEXIT  2002/09/11 Ø1:ØØ:51 *************/
/********** @REFRESH BEGIN MSG      2002/09/11 Ø1:35:53 *************/
/* MSG      - Determine whether to SAY or ISPEXEC SETMSG the message */
/*-------------------------------------------------------------------*/
/* ZEDLMSG  - The long message variable                              */
/*********************************************************************/
 msg: module = 'MSG'
      parse arg zedlmsg
      if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
      parse arg sparms
      push trace() time('L') module 'From:' sigl 'Parms:' sparms
      call modtrace 'START' sigl
```

```
/*********************************************************************/
/* If this is background or OMVS use SAY                             */
/*********************************************************************/
      if tsoenv = 'BACK' | execenv = 'OMVS' then
         say zedlmsg
      else
/*********************************************************************/
/* If this is foreground and ISPF is available, use SETMSG           */
/*********************************************************************/
         do
          if ispfenv = 'YES' then
/*********************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules    */
/*********************************************************************/
            address ISPEXEC "SETMSG MSG(ISRZ000)"
          else
            say zedlmsg
         end
      pull tracelvl . module . sigl . sparms
      call modtrace 'STOP' sigl
      interpret 'trace' tracelvl
      return
/*********** @REFRESH END   MSG      2002/09/11 01:35:53 *************/
/*********** @REFRESH BEGIN DDCHECK  2002/09/11 01:08:30 *************/
/* DDCHECK  - Determine if a required DD is allocated                */
/*-------------------------------------------------------------------*/
/* DD       - DDNAME to confirm                                      */
/*********************************************************************/
 ddcheck: module = 'DDCHECK'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg dd
          dderrmsg = 'OK'
          LRC = listdsi(dd "FILE")
/*********************************************************************/
/* Allow sysreason=3 to verify SYSOUT DD statements                  */
/*********************************************************************/
          if LRC <> 0 & strip(sysreason,'L',0) <> 3 then
             do
              dderrmsg = errmsg(sigl 'Required DD' dd 'is missing')
              call rcexit LRC dderrmsg sysmsglvl2
             end
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return
/*********** @REFRESH END   DDCHECK  2002/09/11 01:08:30 *************/
/*********** @REFRESH BEGIN DDDSNS   2002/09/11 00:37:36 *************/
```

```
/* DDDSNS   - Returns number of DSNs in a DD and populates DDDSNS    */
/*-------------------------------------------------------------------*/
/* TARGDD   - DD to return DSNs for                                  */
/*********************************************************************/
 dddsns: module = 'DDDSNS'
         if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         arg targdd
         if targdd = '' then call rcexit 77 'DD missing for DDDSNS'
/*********************************************************************/
/* Trap the output from the LISTA STATUS command                     */
/*********************************************************************/
         x = outtrap('lines.')
         address TSO "LISTALC STATUS"
         dsnnum = Ø
         ddname = '$DDNAME$'
/*********************************************************************/
/* Parse out the DDNAMEs, locate the target DD and concatentate DSNs */
/*********************************************************************/
         do ddd=1 to lines.Ø
            select
               when words(lines.ddd) = 1 & targdd = ddname &,
                    lines.ddd <> 'KEEP' then
                    dddsns = dddsns strip(lines.ddd)
               when words(lines.ddd) = 1 & strip(lines.ddd),
                    <> 'KEEP' then
                    dddsn.ddd = strip(lines.ddd)
               when words(lines.ddd) = 2 then
                    do
                     parse upper var lines.ddd ddname .
                     if targdd = ddname then
                        do
                         fdsn = ddd - 1
                         dddsns = lines.fdsn
                        end
                    end
               otherwise iterate
            end
         end
/*********************************************************************/
/* Get the last DD                                                   */
/*********************************************************************/
         ddnum = ddlist()
         lastdd = word(ddlist,ddnum)
/*********************************************************************/
/* Remove the last DSN from the list if not the last DD or SYSEXEC   */
/*********************************************************************/
         if targdd <> 'SYSEXEC' & targdd <> lastdd then
```

```
            do
             dsnnum = words(dddsns) - 1
             dddsns = subword(dddsns,1,dsnnum)
            end
/*********************************************************************/
/* Return the number of DSN's in the DD                              */
/*********************************************************************/
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return dsnnum
/*********** @REFRESH END   DDDSNS   2002/09/11 00:37:36 *************/
/*********** @REFRESH BEGIN DDLIST   2002/09/11 00:33:19 *************/
/* DDLIST   - Returns number of DD's and populates DDLIST variable   */
/*-------------------------------------------------------------------*/
/* N/A      - None                                                   */
/*********************************************************************/
 ddlist: module = 'DDLIST'
         if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
/*********************************************************************/
/* Trap the output from the LISTA STATUS command                     */
/*********************************************************************/
         x = outtrap('lines.')
         address TSO "LISTALC STATUS"
         ddnum = 0
/*********************************************************************/
/* Parse out the DDNAMEs and concatenate into a list                 */
/*********************************************************************/
         ddlist = ''
         do ddl=1 to lines.0
            if words(lines.ddl) = 2 then
               do
                parse upper var lines.ddl ddname .
                ddlist = ddlist ddname
                ddnum = ddnum + 1
               end
            else
               do
                iterate
               end
         end
/*********************************************************************/
/* Return the number of DD's                                         */
/*********************************************************************/
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
```

```
         return ddnum
/*********** @REFRESH END   DDLIST   2002/09/11 00:33:19 ************/
/*********** @REFRESH BEGIN TSOTRAP  2002/09/11 01:22:43 ************/
/* TSOTRAP  - Capture the output from a TSO command in a stem        */
/*-------------------------------------------------------------------*/
/* VALIDRC  - Optional valid RC, defaults to zero                    */
/* TSOPARM  - Valid TSO command                                      */
/*********************************************************************/
 tsotrap: module = 'TSOTRAP'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg tsoparm
/*********************************************************************/
/* If the optional valid_rc parm is present use it, if not assume 0  */
/*********************************************************************/
          parse var tsoparm valid_rc tso_cmd
          if datatype(valid_rc,'W') = 0 then
             do
              valid_rc = 0
              tso_cmd = tsoparm
             end
          call outtrap 'tsoout.'
          tsoline = sigl
          address TSO tso_cmd
          CRC = RC
/*********************************************************************/
/* If RC = 0 then return                                             */
/*********************************************************************/
          if CRC <= valid_rc then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return CRC
             end
          else
             do
              trapmsg = center(' TSO Command Error Trap ',78,'-')
              terrmsg = errmsg(sigl 'TSO Command:')
/*********************************************************************/
/* If RC <> 0 then format output depending on environment            */
/*********************************************************************/
              if tsoenv = 'BACK' | execenv = 'OMVS' then
                 do
                  say trapmsg
                  do c=1 to tsoout.0
                     say tsoout.c
                  end
```

```
                say trapmsg
                call rcexit CRC terrmsg tso_cmd
               end
             else
/*********************************************************************/
/* If this is foreground and ISPF is available, use the ISPF LOG     */
/*********************************************************************/
               do
                if ispfenv = 'YES' then
                   do
                    zedlmsg = trapmsg
/*********************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules    */
/*********************************************************************/
                    address ISPEXEC "LOG MSG(ISRZØØØ)"
                    do c=1 to tsoout.Ø
                       zedlmsg = tsoout.c
                       address ISPEXEC "LOG MSG(ISRZØØØ)"
                    end
                    zedlmsg = trapmsg
                    address ISPEXEC "LOG MSG(ISRZØØØ)"
                    call rcexit CRC terrmsg tso_cmd,
                        ' see the ISPF Log (Option 7.5) for details'
                   end
                else
                   do
                    say trapmsg
                    do c=1 to tsoout.Ø
                       say tsoout.c
                    end
                    say trapmsg
                    call rcexit CRC terrmsg tso_cmd
                   end
               end
             end
/*********** @REFRESH END   TSOTRAP  2ØØ2/Ø9/11 Ø1:22:43 ************/
/*********** @REFRESH BEGIN SAYDD    2ØØ2/Ø9/11 Ø1:15:54 ************/
/* SAYDD    - Print messages to the requested DD                     */
/*-------------------------------------------------------------------*/
/* MSGDD    - DDNAME to write messages to                            */
/* MSGLINES - number of blank lines to put before and after          */
/* MESSAGE  - Text to write to the MSGDD                             */
/*********************************************************************/
 saydd: module = 'SAYDD'
        if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
        parse arg msgdd msglines message
        if words(msgdd msglines message) < 3 then
```

```
           call rcexit 33 'Missing MSGDD or MSGLINES'
        if datatype(msglines) <> 'NUM' then
           call rcexit 34 'MSGLINES must be numeric'
/*********************************************************************/
/* Confirm the MSGDD exists                                          */
/*********************************************************************/
        call ddcheck msgdd
/*********************************************************************/
/* If a number is provided, add that number of blank lines before    */
/* and after the message                                             */
/*********************************************************************/
        msgb = 1
        if msglines > 0 then
           do msgb=1 to msglines
              msgline.msgb = ' '
           end
        msgline.msgb = date() time() message
        if msglines > 0 then
           do msgt=1 to msglines
              msge = msgt + msgb
              msgline.msge = ' '
           end
/*********************************************************************/
/* Write the contents of the stack to the MSGDD                      */
/*********************************************************************/
        call tsotrap "EXECIO * DISKW" msgdd "(STEM MSGLINE. FINIS"
        drop msgline. msgb msgt msge
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return
/*********** @REFRESH END   SAYDD    2002/09/11 01:15:54 *************/
/*********** @REFRESH BEGIN JOBINFO  2002/09/11 01:12:59 *************/
/* JOBINFO  - Get job related data from control blocks               */
/*------------------------------------------------------------------*/
/* ITEM     - Optional item number desired, default is all          */
/*********************************************************************/
 jobinfo: module = 'JOBINFO'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg item
/*********************************************************************/
/* Chase control blocks                                              */
/*********************************************************************/
          tcb      = ptr(540)
          ascb     = ptr(548)
          tiot     = ptr(tcb+12)
          jscb     = ptr(tcb+180)
```

```
          ssib     = ptr(jscb+316)
          asid     = c2d(stg(ascb+36,2))
          jobtype  = stg(ssib+12,3)
          jobnum   = strip(stg(ssib+15,5),'L',Ø)
          stepname = stg(tiot+8,8)
          procstep = stg(tiot+16,8)
          program  = stg(jscb+36Ø,8)
          jobdata  = jobtype jobnum stepname procstep program asid
/*******************************************************************/
/* Return job data                                                 */
/*******************************************************************/
          if item <> '' & (datatype(item,'W') = 1) then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return word(jobdata,item)
             end
          else
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return jobdata
             end
/*********** @REFRESH END   JOBINFO  2ØØ2/Ø9/11 Ø1:12:59 *************/
/*********** @REFRESH BEGIN PTR      2ØØ2/Ø7/13 15:45:36 *************/
/* PTR      - Pointer to a storage location                        */
/*-----------------------------------------------------------------*/
/* ARG(1)   - Storage Address                                      */
/*******************************************************************/
 ptr: return c2d(storage(d2x(arg(1)),4))
/*********** @REFRESH END   PTR      2ØØ2/Ø7/13 15:45:36 *************/
/*********** @REFRESH BEGIN STG      2ØØ2/Ø7/13 15:49:12 *************/
/* STG      - Return the data from a storage location              */
/*-----------------------------------------------------------------*/
/* ARG(1)   - Location                                             */
/* ARG(2)   - Length                                               */
/*******************************************************************/
 stg: return storage(d2x(arg(1)),arg(2))
/*********** @REFRESH END   STG      2ØØ2/Ø7/13 15:49:12 *************/
/*********** @REFRESH BEGIN CMDTIMER 2ØØ2/Ø9/11 Ø1:Ø5:32 *************/
/* CMDTIMER - Time a command, if this is a background run          */
/*-----------------------------------------------------------------*/
/* TIMECMD  - Command to time                                      */
/*******************************************************************/
 cmdtimer: module = 'CMDTIMER'
           if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
           parse arg sparms
           push trace() time('L') module 'From:' sigl 'Parms:' sparms
```

```
            call modtrace 'START' sigl
            parse arg timecmd
            if tsoenv = 'BACK' then
               do
                cmdstart = time('e')
                call saydd msgdd Ø 'Starting Command:' timecmd
               end
            interpret timecmd
            CMDRC = RC
            if tsoenv = 'BACK' then
               do
                cmdend = time('e')
                cmddur = format((cmdend - cmdstart),,3)
                call saydd msgdd Ø 'Ending Command:' timecmd', RC='CMDRC
                call saydd msgdd Ø 'Command Duration' cmddur timecmd
               end
            pull tracelvl . module . sigl . sparms
            call modtrace 'STOP' sigl
            interpret 'trace' tracelvl
            return CMDRC
/********** @REFRESH END   CMDTIMER 2ØØ2/Ø9/11 Ø1:Ø5:32 ************/
/********** @REFRESH BEGIN VIODD    2ØØ2/Ø9/11 Ø1:25:Ø1 ************/
/* VIODD    - EXECIO a stem into a TEMP PDS                       */
/*--------------------------------------------------------------*/
/* VIODD    - The member to create                                */
/* VIOLRECL - The LRECL for the VIODD (defautls to 8Ø)            */
/****************************************************************/
 viodd: module = 'VIODD'
        if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
        arg viodd violrecl viorecfm
        if viodd = '' then call rcexit 88 'VIODD missing'
        if violrecl = '' then violrecl = 8Ø
        if viorecfm = '' then viorecfm = 'F B'
/****************************************************************/
/* If DD exists, FREE it                                        */
/****************************************************************/
        if listdsi(viodd 'FILE') = Ø then
            call tsotrap "FREE F("viodd")"
/****************************************************************/
/* ALLOCATE a temporary SYSIN                                   */
/****************************************************************/
        call tsotrap "ALLOC F("viodd") UNIT(VIO) SPACE(1 5)",
                     "LRECL("violrecl") BLKSIZE(Ø) REUSE",
                     "RECFM("viorecfm") CYLINDERS"
/****************************************************************/
/* Write the generated NDM SYSIN statements to the temporary DSN*/
/****************************************************************/
```

```
        call tsotrap "EXECIO * DISKW" viodd "(STEM" viodd". FINIS"
/*********************************************************************/
/* DROP the stem variable                                            */
/*********************************************************************/
        interpret 'drop' viodd'.'
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return
/*********** @REFRESH END   VIODD    2002/09/11 01:25:01 ************/
/*********** @REFRESH BEGIN USSWRAP  2002/09/11 01:24:46 ************/
/* USSWRAP  - Wrapper for USS API commands                           */
/*-------------------------------------------------------------------*/
/* VALIDRC  - Optional valid RC from the USS command, defaults to 0  */
/* USSPARM  - Valid USS command                                      */
/*********************************************************************/
 usswrap: module = 'USSWRAP'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg ussparm
/*********************************************************************/
/* If the optional valid_rc parm is present use it, if not assume 0  */
/*********************************************************************/
          parse var ussparm valid_rc uss_cmd
          if datatype(valid_rc,'W') = 0 then
             do
              valid_rc = 0
              uss_cmd = ussparm
             end
          address SYSCALL uss_cmd
          URC = RC
/*********************************************************************/
/* If RC = 0 then return                                             */
/*********************************************************************/
          if URC <= valid_rc then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return RETVAL
             end
          else
             do
              uerrmsg = errmsg(sigl 'USS Command:')
              call rcexit URC uerrmsg uss_cmd,
                  'RETVAL='RETVAL 'ERRNO='ERRNO 'ERRNOJR='ERRNOJR
             end
/*********** @REFRESH END   USSWRAP  2002/09/11 01:24:46 ************/
```

```
/*********** @REFRESH BEGIN MODTRACE 2002/09/11 01:46:24 ************/
/* MODTRACE - Module Trace                                          */
/*------------------------------------------------------------------*/
/* TRACETYP - Type of trace entry                                   */
/* SIGLINE  - The line number called from                           */
/********************************************************************/
 modtrace: if modtrace = 'NO' then return
           arg tracetyp sigline
           tracetyp = left(tracetyp,5)
           sigline = left(sigline,5)
/********************************************************************/
/* Adjust MODSPACE for START                                        */
/********************************************************************/
           if tracetyp = 'START' then
              modspace = substr(modspace,1,length(modspace)+1)
/********************************************************************/
/* Set the trace entry                                              */
/********************************************************************/
           traceline = modspace time('L') tracetyp module sigline sparms
/********************************************************************/
/* Adjust MODSPACE for STOP                                         */
/********************************************************************/
           if tracetyp = 'STOP' then
              modspace = substr(modspace,1,length(modspace)-1)
/********************************************************************/
/* Determine where to write the traceline                           */
/********************************************************************/
           if ispfenv = 'YES' then
/********************************************************************/
/* Write to the ISPF Log, do not use ISPWRAP here                   */
/********************************************************************/
              do
               zedlmsg = traceline
               address ISPEXEC "LOG MSG(ISRZ000)"
              end
           else
              say traceline
/********************************************************************/
/* SAY to SYSTSPRT                                                  */
/********************************************************************/
           return
/*********** @REFRESH END   MODTRACE 2002/09/11 01:46:24 ************/
```

Robert Zenuk
Systems Programmer (USA)

Communications Server in batch

Although it's not obvious from the IBM manuals, it's possible to access z/OS Communications Server (CS) in batch jobs, even without running TSO (IKJEFT01). This is worth knowing, because batch is typically the easiest way to regularly and automatically run those tasks that network professionals and system programmers need for clean-up, monitoring, and other administrative functions. This article looks at useful ways to access CS functionality in those batch jobs.

CALL ME

Both 'Lights out' and less drastic means of reducing Operations staff through Automated Operations have made manual procedures impractical – especially having the Console Operator phone you every time there's something you should know about. A popular solution is to have software-initiated e-mail sent to you, especially if you have a cell phone, Personal Digital Assistant (PDA), or pager that can receive e-mail and notify you as soon as it arrives.

Although any modern Automated Operations package has an e-mail notification capability, there are three reasons why you may want to have the batch job generate the e-mail itself:

- That's where the problem occurred or was originally identified.
- There's usually more useful technical information available that should go in the e-mail.
- Time is precious in critical situations, and it shouldn't be spent trying to remotely access the mainframe just so you can read the diagnostic information that could have been in the e-mail message.

Simple Mail Transfer Protocol (SMTP) is the Internet e-mail protocol. It defines how to format the datastream to transmit the text and headers of an e-mail message. To send e-mail, CS provides the SMTPNOTE command in TSO, which you could use in batch by running TSO in batch. But CS also provides a direct batch interface through JES2 and JES3 (JES*x*); you can even query the SMTP delivery queues.

This means your batch job writes output to a SYSOUT dataset that JES*x* passes to CS's SMTP address space. Although the JES*x* SYSOUT class and SMTP address space names can be changed, most installations stick to the IBM-supplied defaults of B and SMTP. The DD name varies, but the batch job's DD statement looks as follows:

```
//SENDMAIL DD SYSOUT=(B,SMTP)
```

That SYSOUT output consists of CS SMTP commands. Four SMTP commands are used to define an e-mail message. All four are required, and must appear in the following order: HELO, MAIL FROM, RCPT TO, and DATA. The third is a bit of an exception, as multiple RCPT TO commands can be specified. Only the first RCPT TO must appear where shown.

HELO

All SMTP commands begin with a four letter 'word'. The HELO command identifies the domain name of the sending host to SMTP. If you're unsure of the TCP/IP host name for the z/OS system, it can be found in the TCPIP.DATA sequential dataset. The default IBM DataSet Name (DSN) is TCPIP.TCPIP.DATA. In it, you'll usually find a HOSTNAME definition. From this sample excerpt from the file, you would code HELO S390 as follows:

```
; TCPIPJOBNAME describes the name of the non OE TCP/IP started task
; address space name.
TCPIPJOBNAME TCPIP
;
; HOSTNAME specifies the TCP host name of this system. If not
; specified, the default is P39Ø as defined in the EZAZSSI proc in
```

```
; SYS1.PROCLIB.
;
HOSTNAME S39Ø
;
```

As the comments indicate, HOSTNAME need not be specified in TCPIP.DATA. But checking SYS1.PROCLIB(EZAZSSI) on the same system revealed no P390 value specified.

The semicolons (“;”) in column 1 of TCPIP.DATA indicate comments, but there are so many, it can make it difficult to find the actual definitions. One solution is to use the EXCLUDE (X) command in the ISPF Editor’s View or Edit mode:

```
X ALL 1 X'5E'
```

This displays only the lines without a semicolon in column 1. X'5E' is used instead of ";" (a semicolon in double quotes) because the semicolon is the default value for ISPF’s Command Delimiter, although this can be changed with Option 0 (zero), ISPF Settings.

FROM/TO

The SMTP MAIL FROM command is mandatory and must appear before any RCPT TO command. Both commands are followed by a colon (‘:’), a less than sign (‘<’), the full path of a single e-mail address and a greater than sign (‘>’). Up to 3,000 RCPT TO commands may be used to direct a single message to 3,000 e-mail addresses.

It makes more sense to use your regular e-mail address as the sender, rather than your e-mail address on the mainframe. After all, how often do you check your mainframe e-mail?

DATA

The DATA command is entered alone on a line. All lines that follow form the body of the message. A single period on a line by itself ends the message body. When DATA is the last SMTP command, the message body can be ended by the end-of-file, eliminating the need for the period-only line.

If any line of the message body is longer than 80 characters, a

continuation character can be used in column 80 to split the line after columns 79, 158, 237, etc, as needed. You have a choice of two continuation characters:

- Less than sign ('<')
- EBCDIC New Line (NL) character with a hexadecimal value of 15.

JCL

To get those SMTP commands and the e-mail message body to SMTP, the z/OS batch job must output them to a SYSOUT dataset that JES*x* then routes to SMTP. As shown earlier, unless the IBM defaults have been changed in your installation, the DD statement is coded SYSOUT=(B,SMTP).

In a batch job, if the e-mail message will always be the same, the simplest way to create it is to use an in-stream dataset and copy it to the SMTP SYSOUT dataset. There are many ways to do the copy: IEBGENER, ICEGENER, IDCAMS REPRO, SyncSort, DFSORT, or ICETOOLS. But the SMTP sequence numbers problem makes a Sort/Merge package the best choice.

SEQUENCE NUMBERS

CS programs, including SMTP and FTP, are probably the only utilities, compilers, or other programs commonly used in batch that do not, by default, accept ISPF sequence numbers (columns 73-80) in any (dataset that could conceivably come from an) in-stream dataset. This is despite the fact that many people still use sequence numbers in JCL to provide a form of change control; the ISPF Editor codes a version number into the sequence numbers of each changed line.

The presence of sequence numbers is the most common error made when using SMTP in batch. This intolerance no doubt results from the fact that column 80 is SMTP's continuation column.

If your e-mail message never exceeds a 72-character line

length, the solution can be as simple as blanking out columns 73-80. In a Sort/Merge package, use an INREC or OUTREC statement.

BATCH E-MAIL EXAMPLE

Putting the SMTP commands, JCL, and Sort/Merge statements together, a simple batch job would look as follows:

```
//COPY    EXEC  PGM=ICEMAN
//SYSOUT   DD   SYSOUT=*
//SYSIN    DD   *
  INREC FIELDS=(1,72,8X)
  SORT  FIELDS=COPY
//SORTIN   DD   *
HELO S39Ø
MAIL FROM: <Jon.Pearkins@Adiant.com>
RCPT TO:   <Jon.Pearkins@Adiant.com>
DATA
Job CATCHK found ICF catalogue errors!
//SORTOUT  DD   SYSOUT=(B,SMTP)
//
```

Although not shown, there are sequence numbers on each line of this batch job. This could be the entire batch job, triggered by the Automated Operations system when job CATCHK returns a non-zero condition code. Or it could be added to the CATCHK job, along with a COND= parameter or an IF statement.

TRIGGERING THE MESSAGE

To trigger the message on any previous job step returning a non-zero condition code, a COND parameter would be added to the EXEC statement:

```
//COPY    EXEC  PGM=ICEMAN,COND=(Ø,GE)
```

The IF statement is much less confusing. The equivalent would be:

```
//COPYIF IF (RC > Ø) THEN
//COPY    EXEC  PGM=ICEMAN,COND=(Ø,GE)
 .
 .
 .
//SORTOUT ...
```

```
//       ENDIF
//
```

If you want to use the same message for abends or previous job steps not being run, you could expand the IF statement to read as follows:

```
//COPYIF IF (CATCHK.RUN = FALSE OR ABEND OR RC > Ø) THEN
```

You can even check for specific Abend codes:

```
//COPYIF IF (ABENDCC=SØC4) THEN
//COPYIF IF (ABENDCC=UØØØ1) THEN
```

VERB

The fact that the SMTP batch interface is output-only can make debugging difficult. Where do all the responses go, be they error messages, return codes, positive or negative informational feedback, statistics, or even a response to the SMTP HELP command? Well, they all go to the batch SMTP response dataset, but unfortunately this is automatically discarded unless Verbose mode is turned on with the VERB command.

If you add a new line, VERB ON, before the HELO command, SMTP will use the sender's e-mail address specified by the MAIL TO command to try to return the batch SMTP response dataset. But, as the manual states:

> "If an error occurs during the processing of commands over a batch SMTP connection, such as reception of a negative response (with a first digit of 4 or 5), an error report is mailed back to the sender. The sender is determined from the last MAIL FROM command received that was valid. If the sender cannot be determined from a MAIL FROM command, the sender is assumed to be the origination point of the batch SMTP command dataset. The error report mailed to the sender includes the batch SMTP response dataset and the text of the undeliverable mail."

Of course, if CS isn't configured properly, you may never see the batch SMTP response dataset, no matter what you do. And seemingly valid e-mail address values may fail to work on the

MAIL FROM command. You should look for other indications too, such as the situation I encountered recently on a test system, when the TSO command SMTPNOTE returned the following:

```
IKJ56500I COMMAND SMTPNOTE NOT FOUND
```

OTHER SMTP COMMANDS

The rest of the commands request information from SMTP:

- *XPN*. Checks whether the specified mailbox name exists on the local host.
- *HELP*. Lists all the SMTP commands, or lists information on a specified command.

Version	Manual title(s)	Order #	Chapter			
			SMTP	FTP	Telnet	REXEC/ RSH
z/OS	*z/OS Communications Server: IP User's Guide (and Commands)*	SC31-8780	6	5	2	10
OS/390 V2.10	*OS/390 Communications Server: IP User's Guide*	GC31-8514-06	6	5	2	11
OS/390 V2.5-2.9	*OS/390 eNetwork/ SecureWay Communications Server: IP User's Guide*	GC31-8514	4	3	2	8
OS/390 V1.1-2.4	*TCP/IP Version 3 for MVS: User's Guide*	*SC31-7136*	4	3	2	8
MVS/ESA with TCP/ IP V2.2.1	*TCP/IP Version 2 for MVS: Programmer's Reference*	*SC31-6087*	*12*	-	-	-
TCP/IP V2.2.1	*TCP/IP Version 2 for MVS: User's Guide*	SC31-6088	-	4	2	11

Figure 1: IBM manual references

- *NOOP*. Returns a 250 OK response from SMTP.
- *QUEU*. Returns information on queued mail.
- *QUIT*. Never required in batch.
- *RSET*. Resets the SMTP connection.
- *TICK*. Inserts an identifier into the batch SMTP response dataset.
- *VRFY*. Same as EXPN.

For more information, see the IBM manual appropriate for your version of z/OS, OS/390, or TCP/IP (MVS/ESA) in Figure 1; check the SMTP chapter number listed for a section entitled 'SMTP Commands'. IBM manuals are also available on-line at

http://www.ibm.com/servers/eserver/zseries/zos/bkserv

SUBJECT, TO, CC

Among all the SMTP commands, you won't find one to define the Subject field of the e-mail message. In fact, RCPT TO doesn't fill in the To field, presumably because the e-mail address could be a CC or BCC entry. SMTP determines all these fields from the body of the message, and will even override the From field obtained from the MAIL FROM command, though the MAIL FROM value will still appear in the Return Path field of the Internet mail header.

Revising the previous example, the SMTP command and datastream would look as follows:

```
HELO S390
MAIL FROM: <Jon.Pearkins@Adiant.com>
RCPT TO:   <Jon.Pearkins@Adiant.com>
DATA
From: System Administrator <Jon.Pearkins@Adiant.com>
To:   Jon E. Pearkins <Jon.Pearkins@Adiant.com>
Subject: CATCHK Problem
Job CATCHK found ICF catalogue errors!
```

ATTACHMENTS

As its name states, SMTP is a Simple protocol that doesn't include support for attachments, encryption, or non-printable text in e-mail – other Internet protocols address those needs. However, you can dynamically generate data for an e-mail message in z/OS batch, as in the following example:

```
//COPY    EXEC PGM=ICEMAN
//SYSOUT   DD  SYSOUT=*
//SYSIN    DD  *
  INREC  FIELDS=(1,72,8X)
  SORT   FIELDS=COPY
//SORTIN   DD  *
HELO S390
MAIL FROM: <Jon.Pearkins@Adiant.com>
RCPT TO:   <Jon.Pearkins@Adiant.com>
DATA
From: System Administrator <Jon.Pearkins@Adiant.com>
To:   Jon E. Pearkins <Jon.Pearkins@Adiant.com>
Subject: CATCHK Problem
Job CATCHK found ICF catalogue errors!
//SORTOUT  DD  DSN=&&FB80,SPACE=(TRK,(2,5)),
//             RECFM=FB,LRECL=80,
//             DISP=(NEW,PASS)
//LMCAT   EXEC PGM=IDCAMS
//SYSPRINT DD  DSN=&&VBA133,SPACE=(TRK,(2,5)),
//             RECFM=VBA,LRECL=137,
//             DISP=(NEW,PASS)
//SYSIN    DD  *
  LISTCAT ENTRIES(CATALOG.OS390.MASTER) ALL
//CONVERT EXEC PGM=ICEMAN
//SYSOUT   DD  SYSOUT=*
//SYSIN    DD  *
  OUTFIL OUTREC=(6,79,1X),CONVERT
  SORT   FIELDS=COPY
//SORTIN   DD  DSN=&&VBA133,DISP=(OLD,DELETE)
//SORTOUT  DD  DSN=&&FB80,DISP=(MOD,PASS)
//SPOOL   EXEC PGM=ICEMAN
//SYSOUT   DD  SYSOUT=*
//SYSIN    DD  *
  SORT   FIELDS=COPY
//SORTIN   DD  DSN=&&FB80,DISP=(OLD,DELETE)
//SORTOUT  DD  SYSOUT=(B,SMTP)
//
```

Four steps are required to add IDCAMS output to the end of an e-mail message because SYSPRINT output from IDCAMS is generated in 139-byte VBA records. DFSORT is used to

convert the variable-length records to 80-byte Fixed Block (FB) format.

If you need to see more than the first 79 characters of each IDCAMS print line, change the OUTFIL in the CONVERT job step to read:

```
OUTFIL OUTREC=(6,79,X'15',/,85,53,27X),CONVERT,VLFILL=C' '
```

This splits each IDCAMS print line into two, using the SMTP continuation character (X'15') in column 80 to convince your e-mail system to display the full 132 width of each line. By DFSORT's method of counting, column 6 is the first character of the print line, with columns 1-4 the length field of each variable-length record, and column 5 the carriage control character. The slash splits the single input record into two output records and VLFILL pads blanks on the end of any short variable-length input records. CONVERT performs the variable-to fixed-length conversion.

Note that this OUTFIL statement wasn't tested in SyncSort, and it won't work in older versions of DFSORT – both the slash and VLFILL were new to DFSORT Release 14, which first became available in September 1998. To determine what Sort/Merge product and version you're currently running, run the following batch job:

```
//VERS    EXEC  PGM=SORT
//SYSOUT   DD   SYSOUT=*
//SORTIN   DD   DUMMY
//SORTOUT  DD   DUMMY
//SYSIN    DD   DUMMY
//
```

Expect a Condition Code of 16, but on SYSOUT you should see something like this:

```
ICE000I 1 - CONTROL STATEMENTS FOR 5740-SM1, DFSORT REL 14.0
ICE010A 0 NO SORT OR MERGE CONTROL STATEMENT
ICE052I 3 END OF DFSORT
```

If the size of the e-mail is a problem, selection logic could be added to the Sort/Merge CONVERT job step. Alternatively, the whole process could be written in REXX, running under TSO in

batch, to isolate just the LISTCAT information required, and even format it into the e-mail message itself.

TRANSFERRING FILES

There are many reasons why you might want to transfer datasets or PDS members from one computer system to another, and ftp is usually the best way to do it. One of your most common needs is probably to share freeware or software you've written yourself, usually as source code, but occasionally as object code or a load module. Assuming your z/OS mainframe is either the source or destination, the other system involved might be another mainframe, a workstation, or a non-mainframe host/server.

If a workstation is involved, it's always best to initiate the ftp transfer from the workstation, as they aren't servers, and aren't set up to receive unexpected ftp requests at any moment of the day. A workstation's IP address is also neither fixed nor predictable in today's DHCP-based networks.

For other mainframes, a straight EBCDIC to EBCDIC transfer is the goal, whether the program is source or machine-readable code. Virtually all non-mainframe host/servers will involve source code only and require translation between EBCDIC and ASCII. However, there are some exceptions, such as cbttape.org, where datasets are converted to TSO XMIT format, zipped in EBCDIC, and transferred in binary.

FTP IN BATCH

CS supports ftp in z/OS batch, beyond the use of the ftp command in batch TSO, though you'll need a keen eye to spot it in the *CS IP User's Guide and Commands* manual. Section 4.9 is entitled 'Submitting ftp requests in batch'. There, you'll find that the TSO FTP command and batch JCL listed below are equivalent:

```
FTP 9.67.112.25
//FTP  EXEC PGM=FTP,PARM='9.67.112.25'
```


FTP commands are read from the INPUT DD name; like SMTP, no sequence numbers are allowed. Server responses are listed on either OUTPUT or SYSPRINT, whichever DD name is present. There are a number of reasons why you might want to include a DD statement for NETRC and store the ftp server's ID and password in a dataset or PDS member rather than at the beginning of the INPUT dataset. For example, if INPUT is an in-stream dataset (DD *), you may not want the password to appear in the JCL (even though there are other ways to prevent it from being printed in the job output). More likely, if there are several batch jobs that use the same ftp ID and password, it would be a lot easier to maintain if a password change meant changing a single NETRC file, not every job's JCL.

Many ftp sites maintain an anonymous log-on capability that eliminates the concern over ID and password. You simply log in with the ID of ANONYMOUS and use your e-mail address as a password.

BETWEEN MAINFRAMES

To transfer a file from another mainframe, in this case a PDS member from another z/OS system, the batch job might be as shown below, without ISPF sequence numbers. NETRC would normally be a common dataset or PDS member with many host entries and shared with other batch jobs using FTP. Alternatively, it might have a DSN of userid.NETRC so it would automatically be used for all on-line ftp sessions under that Userid in TSO or z/OS Unix System Services (USS). Here, simply to show everything in one place, it's an in-stream dataset:

```
//FTPTEST EXEC PGM=FTP,PARM='2Ø9.217.251.162'
//OUTPUT   DD SYSOUT=*
//INPUT    DD *
  CD FB132.DATA
  EBCDIC
  GET SHCMDTRP
  QUIT
//NETRC    DD *
MACHINE 2Ø9.217.251.162 LOGIN JONPE PASSWORD MYPW
//
```

CS contacts the host at IP address 209.217.251.162 (from PARM=) on Port 21, the standard FTP port number, requesting a connection. Once connected, a userid and password are requested and provided (JONPE and MYPW) from NETRC. Log-in is completed and ftp commands are accepted, as entered on DD name INPUT:

- Change Directory (CD) into the JONPE.FB132.DATA PDS.
- Set the transfer type to EBCDIC to do a text transfer (not binary) without any ASCII-EBCDIC conversion.
- Transfer (GET) the PDS member SHCMDTRP to your mainframe.
- Terminate (QUIT) the FTP session.

Here's what you'd see on the OUTPUT DD, routed to JESx SYSOUT:

```
EZA1736I FTP 209.217.251.162
EZA1450I IBM FTP CS V2R10  2000 093 23:39 UTC
EZA1554I Connecting to:  209.217.251.162 port: 21.
220-FTPD1 IBM FTP CS V2R10 at S390, 23:49:16 on 2003-01-13.
220 Connection will close if idle for more than 50 minutes.
EZA1701I >>> USER JONPE
331 Send password please.
EZA1701I >>> PASS
230 JONPE is logged on.  Working directory is "JONPE.".
EZA1460I Command:
EZA1736I   CD FB132.DATA
EZA1701I >>> CWD FB132.DATA
250 "JONPE.FB132.DATA" partitioned data set is working directory
EZA1460I Command:
EZA1736I   EBCDIC
EZA1701I >>> TYPE E
200 Representation type is Ebcdic NonPrint
EZA1460I Command:
EZA1736I   GET SHCMDTRP
EZA1701I >>> PORT 209,217,251,162,8,201
200 Port request OK.
EZA1701I >>> RETR SHCMDTRP
125 Sending data set JONPE.FB132.DATA(SHCMDTRP) FIXrecfm 80
250 Transfer completed successfully.
EZA1617I 904 bytes transferred in 0.005 seconds.
         Transfer rate 180.80 Kbytes/sec.
EZA1460I Command:
EZA1736I   QUIT
```

```
EZA17Ø1I >>> QUIT
221 Quit command received. Goodbye.
```

The SHCMDTRP member ends up as a sequential dataset with a DSN of userid.SHCMDTRP, where 'userid' is the RACF ID under which the batch job was run. Despite the 'FIXrecfm 80' listed in the 125 Sending message, the dataset is created Variable Blocked (VB) with a 256-byte record size (LRECL) and 6233-byte block size (BLKSIZE).

You can choose the DSN on your system by specifying it as the second parameter of the ftp GET statement. Adding '(REPLACE' to the end of the statement allows you to pre-allocate the dataset. If you'd prefer a PDS member, specify the member name instead of the DSN as the second parameter on the GET statement, after having set the local directory to the PDS with the LCD (Local Change Directory) command.

CONNECTING TO A NON-MAINFRAME

Only the transfer type differs when connecting to a non-mainframe. The ftp SYSTEM command can be helpful, as it indicates the operating system being run on the host. In the ftp session above, the mainframe responded to the SYSTEM command with a single line, split for printing below:

```
215 MVS is the operating system of this server.
    FTP Server is running on OS/39Ø UNIX System Services.
```

The non-mainframe host used in the next example responded:

```
215 UNIX Type: L8
```

IBM offers a pair of catalogue utilities as freeware at the following address:

```
http://knowledge.storage.ibm.com/vsam/vsaminfo/downloadutilitys.shtml
```

The batch job to obtain the VVDSFIX tool might look as follows:

```
//ALLOCVV EXEC PGM=IEFBR14
//VVDSFIX  DD  DSN=&SYSUID..VVDSFIX.VER13.TRSD,DISP=(MOD,CATLG),
//             DSORG=PS,
//             LRECL=1Ø24,BLKSIZE=27648,RECFM=FB,SPACE=(CYL,(1,2))
//FTPTEST EXEC PGM=FTP
//OUTPUT   DD  SYSOUT=*
```

```
//INPUT    DD  *
  FTP.SOFTWARE.IBM.COM
  ANONYMOUS
  JON.PEARKINS@ADIANT.COM
  CD s390/mvs/tools
  BIN
  GET VVDSFIX.VER13.TRSD (REPLACE
  QUIT
//
```

Rather than use the PARM= and NETRC DD statement, the site, ID, and password are coded as the first three lines of the INPUT in-stream dataset. Because this Unix ftp server is case-sensitive, the subdirectories (s390/mvs/tools) had to be coded in lower-case, and the file (VVDSFIX.VER13.TRSD) had to be coded as upper-case. BIN sets the transfer type to Binary, ensuring that ASCII to EBCDIC translation doesn't occur, nor does the usual elimination of Nulls and other characters that occurs with text transfers.

As documented on the VVDSFIX Instructions Web page, the VVDSFIX sequential dataset must be pre-allocated, and the REPLACE parameter must be used in the FTP GET statement. Otherwise, the LRECL and other dataset attributes will be incorrect. DISP=MOD was used so that the job could be re-run without deleting the dataset.

The OUTPUT SYSOUT dataset looks like this:

```
EZA1736I FTP
EZA1450I IBM FTP CS V2R10  2000 093 23:39 UTC
EZA1456I Connect to ?
EZA1736I   FTP.SOFTWARE.IBM.COM
EZA1554I Connecting to: dispsd-40-www3.boulder.IBM.COM
         207.25.253.40 port: 21.
220 service.boulder.ibm.com FTP server
    (Version wu-2.6.2(1) Mon Dec 3 15:26:19 MST 2001) ready.
EZA1459I NAME (FTP.SOFTWARE.IBM.COM:JONPE):
EZA1701I >>> USER ANONYMOUS
331 Guest login ok, send your complete e-mail address as password.
EZA1789I PASSWORD:
EZA1701I >>> PASS
230-Please read the file README
230-  it was last modified on Thu Aug  9 08:15:27 2001 - 522 days ago
230 Guest login ok, access restrictions apply.
EZA1460I Command:
EZA1736I   CD s390/mvs/tools
```

```
EZA1701I >>> CWD s390/mvs/tools
250 CWD command successful.
EZA1460I Command:
EZA1736I   BIN
EZA1701I >>> TYPE I
200 Type set to I.
EZA1460I Command:
EZA1736I   GET VVDSFIX.VER13.TRSD (replace
EZA1701I >>> PORT 209,217,251,162,8,203
200 PORT command successful.
EZA1701I >>> RETR VVDSFIX.VER13.TRSD
150 Opening BINARY mode data connection for
    VVDSFIX.VER13.TRSD (14336 bytes).
226 Transfer complete.
EZA1617I 14336 bytes transferred in 2.630 seconds.
         Transfer rate 5.45 Kbytes/sec.
EZA1460I Command:
EZA1736I   QUIT
EZA1701I >>> QUIT
221-You have transferred 14336 bytes in 1 files.
221-Total traffic for this session was 14996 bytes in 1 transfers.
221-Thank you for using the FTP service on service.boulder.ibm.com.
221 Goodbye.
```

TELNET AND REXEC

ftp can't do everything you might want to do, especially on another z/OS host. For example, if you're transferring software that comes with a VSAM data file, ftp can't help you as it can't transfer VSAM files. Your best bet is to REPRO the VSAM file to a sequential dataset and then ftp GET the sequential dataset.

Actually, you would want to:

- LISTCAT the remote VSAM file to determine its attribute, not just for a DEFINE CLUSTER on your local machine but also for the sequential dataset you'll need to create for REPRO on the remote host.
- ALLOC the sequential dataset for REPRO.
- DELETE the sequential dataset before the ALLOC, in case you need to re-run the job.
- REPRO the VSAM file to a sequential dataset.

Fortunately, these are all TSO commands, so there's no need to execute JCL in a batch job on the remote host. What's more, they can all be executed on a single line without any prompts.

Telnet, REXEC, and RSH can execute TSO commands on a (remote) z/OS host. All three can be run (locally) in z/OS batch outside of TSO. But according to the manual: "You cannot use the TELNET command to log on to an MVS host from an existing MVS line mode TELNET session."

That leaves REXEC (Remote EXECute) and RSH (Remote Shell Protocol). They use ports 512 and 514 respectively, which may not be open on the remote site's firewall. REXEC and RSH are similar, but not identical, as the following batch job shows:

```
//REXEC   EXEC PGM=REXEC,
//        PARM='2Ø9.217.251.162 LISTCAT ENTRIES(TEST.VSAM) ALL'
//SYSPRINT DD  SYSOUT=*
//NETRC    DD *
MACHINE 2Ø9.217.251.162 LOGIN JONPE PASSWORD MYPW
//*
//RSH     EXEC PGM=RSH,
//        PARM='/-l JONPE/MYPW 2Ø9.217.251.162 LISTCAT'
//SYSPRINT DD  SYSOUT=*
//
```

RSH doesn't support NETRC, but it does support a RHOSTS.DATA dataset in a different format. Also, in batch only, a slash is required as the first character of the PARM field for RSH. For more information on Telnet, REXEC, and RSH, see the chapters in the IBM manual shown in Figure 1 for your version of CS or TCP/IP.

CONCLUSION

There is, of course, much more that you can do with z/OS Communications Server or its predecessors in a batch job, especially if you include the batch TSO environment, EXEC PGM=IKJEFT01. Not to mention what you can do from on-line TSO.

Perhaps most intriguingly, there's a lot you can do, remotely, even from your workstation. You can, of course, send e-mail to

the CS SMTP server and use the command line ftp and Telnet clients of your Windows NT/2000/XP workstation. But I was surprised to find that my Windows XP Professional workstation has REXEC and RSH command line clients. Code some commonly-used TSO commands as REXEC or RSH commands in batch files, put short-cuts to the batch files on your XP desktop, and you could avoid the tedious cycle of starting your 3270 emulator, logging on, typing the command(s), logging off, and shutting down the emulator.

Jon E Pearkins
(Canada)

Information point – reviews

MVS FORUMS – http://www.mvsforums.com/helpboards

At first glance, networking is conspicuously absent from the new MVS Forums. But digging into the Help Boards barely a month after this site debuted, I unearthed lots of little gems. The TSO and ISPF help board includes 'ftp using REXX', which shows REXX code to use ftp to send and receive a PDS member from another (mainframe or non-mainframe) host. 'Other Technical Topics' has several networking items. 'Download All Members of a PDS to Workstation' addresses an FAQ on mainframe help desks, offering a number of solutions using tools as diverse as IEBPTPCH, DFSORT, FTP, MGET, XMIT, ZIP, and PROMPT OFF. '3270 Emulation' looks at a few cheap and free mainframe terminal emulators. And 'MVS Chat' details an intriguing project to host chat sessions on the mainframe.

To date, there have been 951 posts from 277 registered users. The site has already become a Website of the Week on Xephon's *Mainframe Week*. With this much momentum, the next few months should see a huge growth in content.

This is the main entry point to TechTarget. The search facility is useful, allowing you, by default, to search all of their sites, including the four discussed below. These four were selected as they offer the most on networking. You'll probably want to register (free) if you get serious about any of the sites.

Mainframes – http://www.Search390.com

In the centre of the home page, in the 'Browse links in these categories' section, you'll see a picture of the Site Editor; click on MORE CATEGORIES at the bottom of the section. Here are the three most relevant categories, with the subcategories you'll see for each:

- *Networking and communications*, including client/server, connections and interfaces, network management, SNA, TCP/IP, and Web access.
- *Web enabling the mainframe*, including Web application servers, e-commerce/e-business options, Web interoperability, data access, and mainframe as Web host.
- *Downloads*, including IBM software downloads, IBM server downloads, storage systems, and non-IBM sites.

Here, as in all parts of the TechTarget Network, subcategories vary widely as to the number of:

- Custom-written TechTarget material versus links to other sites.
- Links that are out of date, ie material no longer available on the external Web site.

Networking – http://www.SearchNetworking.com

The home page is divided up into a number of sections, including: featured topic, categories, today's top news, what's new, product and vendors – a buyer's guide, white papers, and discussion forums.

As with Search390, click on MORE CATEGORIES, but be aware that the categories here aren't platform-specific. They include: Cisco Press Resource Centre, tutorials, security, network design, installation and configuration, network and systems management, standards and protocols, and storage networks.

Web Services – http://www.SearchWebServices.com

The home page layout is similar to networking, with the MORE CATEGORIES link leading you to a mostly non-platform-specific list including: Web Services architect, Web Services basics, SOAP, UDDI, WSDL, XML, and related protocols and APIs.

What Is? – http://www.WhatIs.com

More than a dictionary, whatis?com, as it likes to be known, has encyclopaedia-length articles for each computing term, links to additional material, and the subcategory where the term fits in TechTarget.

Jon E Pearkins
(Canada)

E-mail alerts

Our e-mail alert service will notify you when new issues of *TCP/SNA Update* have been placed on our Web site. If you'd like to sign up, go to http://www.xephon.com/tcpsna and click the 'Receive an e-mail alert' link.

March 1997 – March 2003 index

Items below are references to articles that have appeared in *TCP/SNA Update* since March 1996. 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.

Interested in writing an article, but not sure what on?

We've been asked to commission articles on a variety of TCP/SNA-related topics. Visit the *TCP/SNA Update* Web site, http://www.xephon.com/tcpsna, and follow the link to Opportunities for TCP/SNA specialists.

WRQ has made a number of announcements:

- WRQ Verastream Host Integrator 5.0's Design Tool now allows developers to mix and match between 3270 screens that are automatically converted to Web pages and presented 'as is', and abstraction of legacy functionality to make mainframe applications more intuitive.
- WRQ Reflection 10.0 fully integrates VBA 6.3 and adds an administrator's toolkit, an integrated OpenSSH client, and a custom interface that displays Reflection with a same look and feel as Windows XP, and provides interoperability with WRQ Reflection for the Web through settings conversion capabilities between the two products.

URLs:
http://www.wrq.com/aboutwrq/news/2002/102102pr.html
http://www.wrq.com/aboutwrq/news/2002/092502apr.html

* * *

IBM and Tivoli have made a number of announcements:

- IBM's new Application Workload Modeller can be run on OS/390 2.10, z/OS (202-338) or mainframe Linux (203-001) to determine the network impact of an application before it's written.
- Session Manager for z/OS (202-349) provides simultaneous access to multiple terminal sessions through TCP/IP or VTAM. Datastream optimization and a performance monitor are also included.
- Tivoli Monitoring for Network Performance (202-278) combines IBM Tivoli NetView Performance Monitor 2.7 (202-008) and IBM Tivoli NetView for TCP/IP Performance 1.4 (202-083) into a single package.
- Tivoli Remote Control 3.8 (202-350) can be run on SuSE Linux 7.0 for System/390, adds support for Windows XP, has a firewall friendly architecture, provides full datastream encryption, and offers a new Web interface.
- Tivoli System Automation for OS/390 (SA OS/390) 2.2 (202-256) adds network automation support for registering an application with VTAM application node recovery, issuing recovery commands for all registered applications, and listing by applications major nodes that are in use.
- Tivoli Switch Analyzer 1.2 (202-254) now supports Linux for zSeries.
- Tivoli Monitoring for Transaction Performance 5.1 (202-303) adds support for transaction performance monitoring of SAP and 3270 applications through transaction simulation.

URL: http://www.ibmlink.ibm.com/usalets&parms=H_*nnn-nnn*, where '*nnn-nnn*' is one of the numbers in brackets shown above.

* * *

Software Diversified Services (SDS) has released Vital Signs VisionNet (VSV) 5.1.0, offering complete z/OS network management and performance monitoring through a new Web browser interface.

URL: http://www.sdsusa.com/vsv

* * *