

Network Working Group  
Request for Comments: 4777  
Obsoletes: 2877  
Category: Informational

T. Murphy, Jr.  
P. Rieth  
J. Stevens  
IBM  
November 2006

## IBM's iSeries Telnet Enhancements

### Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

### Copyright Notice

Copyright (C) The IETF Trust (2006).

### IESG Note

This RFC is not a candidate for any level of Internet Standard. The IETF disclaims any knowledge of the fitness of this RFC for any purpose and in particular notes that the decision to publish is not based on IETF review for such things as security, congestion control, or inappropriate interaction with deployed protocols. The RFC Editor has chosen to publish this document at its discretion. Readers of this document should exercise caution in evaluating its value for implementation and deployment. See RFC 3932 for more information.

### Abstract

This document describes the interface to the Telnet server on IBM's iSeries line of midrange business computers. This interface allows Telnet clients to request a Telnet terminal or printer session using specific session attributes related to device names, encryption, language support, auto-sign-on, response codes, session association, etc.

These support functions are implemented primarily using the Telnet Environment option negotiation RFC 1572 to define new USERVAR variables that will be recognized by iSeries Telnet server.

## Table of Contents

1. Introduction .....	2
2. Standard Telnet Option Negotiation .....	3
3. Enhanced Telnet Option Negotiation .....	4
4. Enhanced Display Emulation Support .....	7
5. Enhanced Display Auto-Sign-On and Password Encryption .....	9
5.1. Data Encryption Standard (DES) Password Substitutes .....	13
5.2. Secure Hash Algorithm (SHA) Password Substitutes .....	16
6. Kerberos Services Ticket Automatic Sign-On Support .....	18
7. Device Name Collision Processing .....	21
8. Enhanced Printer Emulation Support .....	22
9. Telnet Printer Terminal Types .....	23
10. Startup Response Record for Printer and Display Devices .....	25
10.1. Example of a Success Response Record .....	26
10.2. Example of an Error Response Record .....	27
10.3. Example of a Response Record with Device Name Retry .....	28
10.4. Response Codes .....	31
11. Printer Steady-State Pass-Through Interface .....	33
11.1. Example of a Print Record .....	35
11.2. Example of a Print Complete Record .....	37
11.3. Example of a Null Print Record .....	37
12. End-to-End Print Example .....	39
13. Security Considerations .....	44
14. IANA Considerations .....	44
15. Normative References .....	44
16. Informative References .....	44
17. Relation to Other RFCs .....	45

## 1. Introduction

The iSeries Telnet server enables clients to negotiate both terminal and printer device names through Telnet Environment Options Negotiations [RFC1572].

This allows Telnet servers and clients to exchange environment information using a set of standard or custom variables. By using a combination of both standard VARs and custom USERVARs, the iSeries Telnet server allows client Telnet to request a pre-defined specific device by name.

If no pre-defined device exists, then the device will be created, with client Telnet having the option to negotiate device attributes, such as the code page, character set, keyboard type, etc.

Since printers can now be negotiated as a device name, new terminal types have been defined to request printers. For example, you can

now negotiate "IBM-3812-1" and "IBM-5553-B01" as valid `TERMINAL-TYPE` options [RFC1091].

Finally, the iSeries Telnet server will allow exchange of user profile and password information, where the password may be in either plain text or encrypted form. If a valid combination of profile and password is received, then the client is allowed to bypass the sign-on panel. The local server setting of the `QRMTSIGN` system value must be either `*VERIFY` or `*SAMEPRF` for the bypass of the sign-on panel to succeed.

## 2. Standard Telnet Option Negotiation

Telnet server option negotiation [RFC855] typically begins with the issuance, by the server, of an invitation to engage in terminal type negotiation with the Telnet client (`DO TERMINAL-TYPE`) [RFC1091]. The client and server then enter into a series of sub-negotiations to determine the level of terminal support that will be used. After the terminal type is agreed upon, the client and server will normally negotiate a required set of additional options (`EOR` [RFC885], `BINARY` [RFC856], `SGA` [RFC858]) that are required to support "transparent mode" or full screen 5250/3270 block mode support. As soon as the required options have been negotiated, the server will suspend further negotiations and begin with initializing the actual virtual device on the iSeries. A typical exchange might start as follows:

iSeries Telnet server		Enhanced Telnet client
-----		-----
IAC DO TERMINAL-TYPE	-->	
	<--	IAC WILL TERMINAL-TYPE
IAC SB TERMINAL-TYPE SEND		
IAC SE	-->	
	<--	IAC SB TERMINAL-TYPE IS
		IBM-5555-C01 IAC SE
IAC DO EOR	-->	
	<--	IAC WILL EOR
	<--	IAC DO EOR
IAC WILL EOR	-->	
	.	
	.	
(other negotiations)	.	

Actual bytes transmitted in the above example are shown in hex below.

iSeries Telnet server		Enhanced Telnet client
-----		-----
FF FD 18	-->	
	<--	FF FB 18
FF FA 18 01 FF F0	-->	
		FF FA 18 00 49 42 4D 2D
		35 35 35 35 2D 43 30 31
	<--	FF F0
FF FD 19	-->	
	<--	FF FB 19
	<--	FF FD 19
FF FB 19	-->	
	.	
	.	
(other negotiations)	.	

Some negotiations are symmetrical between client and server, and some are negotiated in one direction only. Also, it is permissible and common practice to bundle more than one response or request, or to combine a request with a response, so in practice the actual exchange may look different from what is shown above.

### 3. Enhanced Telnet Option Negotiation

In order to accommodate the new environment option negotiations, the server will bundle an environment option invitation along with the standard terminal type invitation request to the client.

A client should either send a negative acknowledgment (WONT NEW-ENVIRON), or at some point after completing terminal-type negotiations, but before completing the full set of negotiations required for 5250 transparent mode, engage in environment option sub-negotiation with the server. A maximum of 1024 bytes of environment strings may be sent to the server. A recommended sequence might look like the following:

iSeries Telnet server		Enhanced Telnet client
-----		-----
IAC DO NEW-ENVIRON		
IAC DO TERMINAL-TYPE	-->	
(2 requests bundled)		
	<--	IAC WILL NEW-ENVIRON
IAC SB NEW-ENVIRON SEND		
VAR IAC SE	-->	
		IAC SB NEW-ENVIRON IS
		VAR "USER" VALUE "JONES"
		USERVAR "DEVNAME"
		VALUE "MYDEVICE07"
	<--	IAC SE
	<--	IAC WILL TERMINAL-TYPE
		(do the terminal type
		sequence first)
IAC SB TERMINAL-TYPE SEND		
IAC SE	-->	
		IAC SB TERMINAL-TYPE IS
	<--	IBM-5555-C01 IAC SE
		(terminal type negotiations
		completed)
IAC DO EOR	-->	
(server will continue		
with normal transparent		
mode negotiations)		
	<--	IAC WILL EOR
	.	
	.	
(other negotiations)	.	

Actual bytes transmitted in the above example are shown in hex below.

iSeries Telnet server		Enhanced Telnet client
-----		-----
FF FD 27		
FF FD 18	-->	
(2 requests bundled)		
	<--	FF FB 27
FF FA 27 01 00 FF F0	-->	
		FF FA 27 00 00 55 53 45
		52 01 4A 4F 4E 45 53 03
		44 45 56 4E 41 4D 45 01
		4D 59 44 45 56 49 43 45
	<--	30 37 FF F0
	<--	FF FB 18
		(do the terminal type
		sequence first)
FF FA 18 01 FF F0	-->	
		FF FA 18 00 49 42 4D 2D
		35 35 35 35 2D 43 30 31
	<--	FF F0
FF FD 19	-->	
(server will continue		
with normal transparent		
mode negotiations)		
	<--	FF FB 19
		.
		.
(other negotiations)		.

Telnet environment options defines 6 standard VARs: USER, JOB, ACCT, PRINTER, SYSTEMTYPE, and DISPLAY. The USER standard VAR will hold the value of the iSeries user profile name to be used in auto-sign-on requests. The Telnet server will make no direct use of the additional 5 VARs, nor are any of them required to be sent. All standard VARs and their values that are received by the Telnet server will be placed in a buffer, along with any USERVARs received (described below), and made available to a registered initialization exit program to be used for any purpose desired.

There are some reasons you may want to send NEW-ENVIRON negotiations prior to TERMINAL-TYPE negotiations. With an iSeries Telnet server, several virtual device modes can be negotiated: 1) VTxxx device, 2) 3270 device, and 3) 5250 device (includes Network Station). The virtual device mode selected depends on the TERMINAL-TYPE negotiated plus any other Telnet option negotiations necessary to support those modes. The iSeries Telnet server will create the desired virtual device at the first opportunity it thinks it has all the requested

attributes needed to create the device. This can be as early as completion of the `TERMINAL-TYPE` negotiations.

For the case of Transparent mode (5250 device), the moment `TERMINAL-TYPE`, `BINARY`, and `EOR` options are negotiated, the Telnet server will go create the virtual device. Receiving any `NEW-ENVIRON` negotiations after these option negotiations are complete will result in the `NEW-ENVIRON` negotiations having no effect on device attributes, as the virtual device will have already been created.

So, for Transparent mode, `NEW-ENVIRON` negotiations are effectively closed once `EOR` is negotiated, since `EOR` is generally the last option done.

For other devices modes (such as `VTxxx` or `3270`), you cannot be sure when the iSeries Telnet server thinks it has all the attributes to create the device. Recall that `NEW-ENVIRON` negotiations are optional, and therefore the iSeries Telnet server need not wait for any `NEW-ENVIRON` options prior to creating the virtual device. It is in the clients' best interest to send `NEW-ENVIRON` negotiations as soon as possible, preferably before `TERMINAL-TYPE` is negotiated. That way, the client can be sure that the requested attributes were received before the virtual device is created.

#### 4. Enhanced Display Emulation Support

Telnet environment option `USERVARs` have been defined to allow a compliant Telnet client more control over the Telnet server virtual device on the iSeries and to provide information to the Telnet server about the client. These `USERVARs` allow the client Telnet to create or select a previously created virtual device. If the virtual device does not exist and must be created, then the `USERVAR` variables are used to create and initialize the device attributes. If the virtual device already exists, the device attributes are modified.

The USERVARs defined to accomplish this are:

USERVAR	VALUE	EXAMPLE	DESCRIPTION
-----	-----	-----	-----
DEVNAME	us-ascii char(x)	MYDEVICE07	Display device name
KBDTYPE	us-ascii char(3)	USB	Keyboard type
CODEPAGE	us-ascii char(y)	437	Code page
CHARSET	us-ascii char(y)	1212	Character set
IBMSENDCONFREC	us-ascii char(3)	YES   NO	Startup Response Record desired
IBMASSOCPRT	us_ascii char(x)	RFCPRT	Printer associated with display device

x - up to a maximum of 10 characters

y - up to a maximum of 5 characters

For a description of the KBDTYPE, CODEPAGE, and CHARSET parameters and their permissible values, refer to Chapter 8 in the Communications Configuration Reference [COMM-CONFIG] and also to Appendix C in National Language Support [NLS-SUPPORT].

The CODEPAGE and CHARSET USERVARs must be associated with a KBDTYPE USERVAR. If either CODEPAGE or CHARSET are sent without KBDTYPE, they will default to system values. A default value for KBDTYPE can be sent to force CODEPAGE and CHARSET values to be used.

iSeries system objects such as device names, user profiles, plain text passwords, programs, libraries, etc., are required to be specified in English uppercase. This includes:

any letter (A-Z), any number (0-9), special characters (# \$ \_ @)

Therefore, where us-ascii is specified for VAR or USERVAR values, it is recommended that uppercase ASCII values be sent, which will be converted to Extended Binary Coded Decimal Interchange Code (EBCDIC) by the Telnet server.

A special case occurs for encrypted passwords (described in the next section), where both the initial password and user profile used to build the encrypted password must be EBCDIC English uppercase, in order to be properly authenticated by the Telnet server.

The IBMASSOCPRT USERVAR is used to provide the device name of a printer that will be associated with the display device that is created. The device description of the printer name provided must currently exist on the Telnet server system. The IBMSENDCONFREC USERVAR is used by the enhanced Telnet client to inform the Telnet



server that a display Startup Response Record should be sent to the client. This record communicates the name of the actual display device acquired. If the attempt is unsuccessful, the reason code will be set to provide additional information on why the attempt failed. In addition to the device name and reason code, the Startup Response Record will contain the name of the Telnet server system.

For more details on the Startup Response Record, see Section 11 of this document.

## 5. Enhanced Display Auto-Sign-On and Password Encryption

To allow password encryption, new IBMRSEED and IBMSUBSPW USERVARs will be used to exchange seed and substitute passwords information. IBMRSEED will carry a random seed to be used in both the Data Encryption Standard (DES) and Secure Hash Algorithm (SHA) password encryption, and IBMSUBSPW will carry the encrypted copy of the password.

The DES encryption would use the same 7-step DES-based password substitution scheme as APPC and Client Access. For a description of

DES encryption, refer to Federal Information Processing Standards Publications (FIPS) 46-2 [FIPS-46-2] and 81 [FIPS-81].

The SHA encryption is described in Federal Information Processing Standards Publication 180-1 [FIPS-180-1].

The FIPS documents can be found at the Federal Information Processing Standards Publications link:

<http://www.itl.nist.gov/fipspubs/by-num.htm>

If encrypted password exchange is not required, plain text password exchange is permitted using the same USERVARs defined for encryption. For this case, the random client seed should be set either to an empty value (preferred method) or to hexadecimal zeros to indicate the password is not encrypted, but is plain text.

It should be noted that security of plain text password exchange cannot be guaranteed unless the network is physically protected or a trusted network (such as an intranet). If your network is vulnerable to IP address spoofing or directly connected to the Internet, you should engage in encrypted password exchange to validate a client's identity.

Additional VARs and USERVARs have also been defined to allow an auto-sign-on user greater control over their startup environment,

similar to what is supported using the Open Virtual Terminal (QTVOPNVT) API [SYSTEM-API].

The standard VARs supported to accomplish this are:

VAR	VALUE	EXAMPLE	DESCRIPTION
-----	-----	-----	-----
USER	us-ascii char(x)	USERXYZ	User profile name

x - up to a maximum of 10 characters

The custom USERVARs defined to accomplish this are:

USERVAR	VALUE	EXAMPLE	DESCRIPTION
-----	-----	-----	-----
IBMRSEED	binary(8)	8-byte hex field	Random client seed
IBMSUBSPW	binary(128)	128-byte hex field	Substitute password
IBMCURLIB	us-ascii char(x)	QGPL	Current library
IBMIMENU	us-ascii char(x)	MAIN	Initial menu
IBMPROGRAM	us-ascii char(x)	QCMD	Program to call

x - up to a maximum of 10 characters

In order to communicate the server random seed value to the client, the server will request a USERVAR name made up of a fixed part (the 8 characters "IBMRSEED") immediately followed by an 8-byte hexadecimal variable part, which is the server random seed. The client generates its own 8-byte random seed value and uses both seeds to encrypt the password. Both the encrypted password and the client random seed value are then sent to the server for authentication. Telnet environment option rules will need to be adhered to when transmitting the client random seed and substituted password values to the server. Specifically, since a typical environment string is a variable length hexadecimal field, the hexadecimal fields are required to be escaped and/or byte stuffed according to the RFC 854 [RFC854], where any single byte could be misconstrued as a Telnet IAC or other Telnet option negotiation control character. The client must escape and/or byte stuff any bytes that could be seen as a Telnet environment option, specifically VAR, VALUE, ESC, and USERVAR.

If you use the IBMSENDCONFREC USERVAR, as described in Section 5 of this document, with a value of YES along with the automatic sign-on USERVARs described above, you will receive a Startup Response Record that will contain a response code informing your Telnet client of the success or failure of the automatic sign-on attempt. See Section 11 of this document for details on the Startup Response Record.

The following illustrates the encrypted case:

iSeries Telnet server		Enhanced Telnet client
-----		-----
IAC DO NEW-ENVIRON	-->	
	<--	IAC WILL NEW-ENVIRON
IAC SB NEW-ENVIRON SEND		
USERVAR "IBMRSEEDxxxxxxxx"		
USERVAR "IBMSUBSPW"		
VAR USERVAR IAC SE	-->	IAC SB NEW-ENVIRON IS
		VAR "USER" VALUE "DUMMYUSR"
		USERVAR "IBMRSEED" VALUE "YYYYYYYYY"
		USERVAR "IBMSUBSPW" VALUE "zzzzzzzzz"
	<--	IAC SE
	.	
	.	
(other negotiations)	.	

In this example, "xxxxxxxx" is an 8-byte hexadecimal random server seed, "YYYYYYYYY" is an 8-byte hexadecimal random client seed, and "zzzzzzzzz" is an 8-byte hexadecimal encrypted password (if the DES encryption algorithm was used) or a 20-byte hexadecimal encrypted password (if the SHA encryption algorithm was used). If the password is not valid, then the sign-on panel is not bypassed. If the password is expired, then the sign-on panel is not bypassed.

Actual bytes transmitted in the above example are shown in hex below, where the server seed is "7D3E488F18080404", the client seed is "4E4142334E414233", and the DES encrypted password is "DFB0402F22ABA3BA". The user profile used to generate the encrypted password is "44554D4D59555352" (DUMMYUSR), with a plain text password of "44554D4D595057" (DUMMYPW).

iSeries Telnet server		Enhanced Telnet client
-----		-----
FF FD 27	-->	
	<--	FF FB 27
FF FA 27 01 03 49 42 4D		
52 53 45 45 44 7D 3E 48		
8F 18 08 04 04 03 49 42		
4D 53 55 42 53 50 57 03		
00 FF F0	-->	
		FF FA 27 00 00 55 53 45
		52 01 44 55 4D 4D 59 55
		53 52 03 49 42 4D 52 53
		45 45 44 01 4E 41 42 33
		4E 41 42 33 03 49 42 4D
		53 55 42 53 50 57 01 DF
		B0 40 2F 22 AB A3 BA FF
	<--	F0

The following illustrates the plain text case:

iSeries Telnet server		Enhanced Telnet client
-----		-----
IAC DO NEW-ENVIRON	-->	
	<--	IAC WILL NEW-ENVIRON
IAC SB NEW-ENVIRON SEND		
USERVAR "IBMRSEEDxxxxxxxx"		
USERVAR "IBMSUBSPW"		
VAR USERVAR IAC SE	-->	
		IAC SB NEW-ENVIRON IS
		VAR "USER" VALUE "DUMMYUSR"
		USERVAR "IBMRSEED" VALUE
		USERVAR "IBMSUBSPW" VALUE "YYYYYYYY"
	<--	IAC SE
	.	
	.	
(other negotiations)	.	

In this example, "xxxxxxxx" is an 8-byte hexadecimal random server seed, and "YYYYYYYYYY" is a 128-byte us-ascii client plain text password. If the password has expired, then the sign-on panel is not bypassed.

Actual bytes transmitted in the above example are shown in hex below, where the server seed is "7D3E488F18080404", the client seed is empty, and the plain text password is "44554D4D595057" (DUMMYPW). The user profile used is "44554D4D59555352" (DUMMYUSR).

iSeries Telnet server		Enhanced Telnet client
-----		-----
FF FD 27	-->	
	<--	FF FB 27
FF FA 27 01 03 49 42 4D		
52 53 45 45 44 7D 3E 48		
8F 18 08 04 04 03 49 42		
4D 53 55 42 53 50 57 03		
00 FF F0	-->	
		FF FA 27 00 00 55 53 45
		52 01 44 55 4D 4D 59 55
		53 52 03 49 42 4D 52 53
		45 45 44 01 03 49 42 4D
		53 55 42 53 50 57 01 44
	<--	55 4D 4D 59 50 57 FF F0

### 5.1. Data Encryption Standard (DES) Password Substitutes

Both APPC and Client Access use well-known DES encryption algorithms to create encrypted passwords. A Network Station or Enhanced Client can generate compatible encrypted passwords if it follows these steps, details of which can be found in the Federal Information Processing Standards 46-2 [FIPS-46-2].

- 1) Padded\_PW = Left justified user password padded to the right with '40'X to 8 bytes.

The user's password must be left justified in an 8-byte variable and padded to the right with '40'X up to an 8-byte length. If the user's password is 8 bytes in length, no padding will occur. For computing password substitutes for passwords of length 9 and 10, see "Handling passwords of length 9 and 10" below. Passwords less than 1 byte or greater than 10 bytes in length are not valid. Please note that if password is not in EBCDIC, it must be converted to EBCDIC uppercase.

- 2) XOR\_PW = Padded\_PW xor '5555555555555555'X

The padded password is Exclusive OR'ed with 8 bytes of '55'X.

- 3) SHIFT\_RESULT = XOR\_PW << 1

The entire 8-byte result is shifted 1 bit to the left; the leftmost bit value is discarded, and the rightmost bit value is cleared to 0.

- 4) PW\_TOKEN = DES\_ECB\_mode(SHIFT\_RESULT, /\* key \*/  
                              userID\_in\_EBCDIC\_uppercase /\* data \*/ )

This shifted result is used as key to the Data Encryption Standard (Federal Information Processing Standards 46-2 [FIPS-46-2]) to encipher the user identifier. When the user identifier is less than 8 bytes, it is left justified in an 8-byte variable and padded to the right with '40'X. When the user identifier is 9 or 10 bytes, it is first padded to the right with '40'X to a length of 10 bytes. Then bytes 9 and 10 are "folded" into bytes 1-8 using the following algorithm:

Bit 0 is the high-order bit (i.e., has value of '80'X).

Byte 1, bits 0 and 1 are replaced with byte 1, bits 0 and 1 Exclusive OR'ed with byte 9, bits 0 and 1.  
Byte 2, bits 0 and 1 are replaced with byte 2, bits 0 and 1 Exclusive OR'ed with byte 9, bits 2 and 3.  
Byte 3, bits 0 and 1 are replaced with byte 3, bits 0 and 1 Exclusive OR'ed with byte 9, bits 4 and 5.  
Byte 4, bits 0 and 1 are replaced with byte 4, bits 0 and 1 Exclusive OR'ed with byte 9, bits 6 and 7.  
Byte 5, bits 0 and 1 are replaced with byte 5, bits 0 and 1 Exclusive OR'ed with byte 10, bits 0 and 1.  
Byte 6, bits 0 and 1 are replaced with byte 6, bits 0 and 1 Exclusive OR'ed with byte 10, bits 2 and 3.  
Byte 7, bits 0 and 1 are replaced with byte 7, bits 0 and 1 Exclusive OR'ed with byte 10, bits 4 and 5.  
Byte 8, bits 0 and 1 are replaced with byte 8, bits 0 and 1 Exclusive OR'ed with byte 10, bits 6 and 7.

User identifiers greater than 10 bytes or less than 1 byte are not the result of this encryption ID, known as PW\_TOKEN in the paper.

5) Increment PWSEQs and store it.

Each LU must maintain a pair of sequence numbers for ATTACHs sent and received on each session. Each time an ATTACH is generated, (and password substitutes are in use on the session) the sending sequence number, PWSEQs, is incremented and saved for the next time. Both values are set to zero at BIND time. So the first use of PWSEQs has the value of 1 and increases by one with each use. A new field is added to the ATTACH to carry this sequence number. However, in certain error conditions, it is possible for the sending side to increment the sequence number, and the receiver may not increment it. When the sender sends a subsequent ATTACH, the receiver will detect a missing sequence. This is allowed. However the sequence number received must always be larger than the previous one, even if some are missing.

The maximum number of consecutive missing sequence numbers allowed is 16. If this is exceeded, the session is unbound with a protocol violation.

Note: The sequence number must be incremented for every ATTACH sent. However, the sequence number field is only required to be included in the FMH5 if a password substitute is sent (byte 4, bit 3 on).

6) RDrSEQ = RDr + PWSEQs /\* RDr is server seed. \*/

The current value of PWSEQs is added to RDr, the random value received from the partner LU on this session, yielding RDrSEQ, essentially a predictably modified value of the random value received from the partner LU at BIND time.

```
7) PW_SUB = DES_CBC_mode(PW_TOKEN,      /* key      */
                          (RDrSEQ,      /* 8 bytes  */
                           RDs,         /* 8 bytes  */
                           ID xor RDrSEQ, /* 16 bytes */
                           PWSEQs,      /* 8 bytes  */
                           ),           /* data     */
                          )
```

The PW\_TOKEN is used as a key to the DES function to generate an 8-byte value for the following string of inputs. The DES CBC mode Initialization Vector (IV) used is 8 bytes of '00'X.

RDrSEQ: the random data value received from the partner LU plus the sequence number.

RDs: the random data value sent to the partner LU on BIND for this session.

A 16-byte value created by:

- padding the user identifier with '40'X to a length of 16 bytes.
- Exclusive OR'ing the two 8-byte halves of the padded user identifier with the RDrSEQ value.

Note: User ID must first be converted to EBCDIC uppercase.

PWSEQs: the sequence number.

This is similar to the process used on LU-LU verification as described in the Enhanced LU-LU Bind Security. The resulting enciphered random data is the 'password substitute'.

#### 8) Handling passwords of length 9 and 10

1. Generate PW\_TOKENa by using characters 1 to 8 of the password and steps 1-4 from the previous section.
2. Generate PW\_TOKENb by using characters 9 and 10 and steps 1-4 from the previous section. In this case, Padded\_PW from step 1 will be characters 9 and 10 padded to the right with '40'X, for a total length of 8.
3. PW\_TOKEN = PW\_TOKENa xor PW\_TOKENb
4. Now compute PW\_SUB by performing steps 5-7 from the previous section.

#### 9) Example DES Password Substitute Calculation

ID: USER123  
Password: ABCDEFG  
Server seed: '7D4C2319F28004B2'X  
Client seed: '08BEF662D851F4B1'X  
PWSEQs: 1 (PWSEQs is a sequence number needed in the  
7-step encryption, and it is always one)

DES Encrypted Password should be: '5A58BD50E4DD9B5F'X

### 5.2. Secure Hash Algorithm (SHA) Password Substitutes

A Network Station or Enhanced Client can generate SHA encrypted passwords if it follows these steps.

- 1) Convert the user identifier to uppercase UNICODE format (if it is not already in this format).

The user identifier must be left justified in a 10-byte variable and padded to the right with '40'X up to a 10-byte length prior to converting it to UNICODE. If the user's password is 10 bytes in length, no padding will occur. User identifiers of less than 1 byte or greater than 10 bytes in length are not valid. The user identifier will be 20 bytes in length after conversion to UNICODE, so the variable that will hold the UNICODE user identifier should have a length of 20 bytes.



- 2) Ensure the password is in UNICODE format (if it is not already in this format).

The user's password must be left justified in a 128-byte variable. It does not need to be padded to the right with '40'X up to a 128-byte length. Passwords less than 1 byte or greater than 128 bytes in length are not valid. The password will be 2 times its original length after conversion to UNICODE, so the maximum length of the variable that will hold the UNICODE password is 256 bytes.

- 3) Create a 20-byte password token as follows:

```
PW_token = SHA-1(uppercase_unicode_userid,      /* 20 bytes */
                  unicode_password)             /* from 2 to 256 bytes */
```

The actual routine to be used to perform the SHA-1 processing is dependent on the programming language being used. For example, if using the Java language, then use the java.security class to perform the actual SHA-1 processing.

The PW\_token will be used in subsequent step to actually generate the final substitute password.

- 4) Increment PWSEQs and store it.

- 5) Create the 20-byte substitute password as follows:

```
PW_SUB = SHA-1(PW_token,                        /* 20 bytes */
                serverseed,                      /* 8 bytes */
                clientseed,                     /* 8 bytes */
                uppercase_unicode_userid,        /* 20 bytes */
                PWSEQ)                          /* 8 bytes */
```

The actual routine to be used to perform the SHA-1 processing is dependent on the programming language being used. For example, if using the Java language, then use the java.security class to perform the actual SHA-1 processing.

- 6) Example SHA Password Substitute Calculation

```
ID:                USER123
Password:          AbCdEfGh123?+
Server seed:       '3E3A71C78795E5F5'X
Client seed:       'B1C806D5D377D994'X
PWSEQs:            1      (PWSEQs is a sequence number needed in the
                           SHA encryption, and it is always one)
```

SHA Encrypted Password should be:

'E7FAB5F034BEDA42E91F439DD07532A24140E3DD'X

## 6. Kerberos Services Ticket Automatic Sign-On Support

An iSeries Telnet server specific USERVAR defined below will contain the complete Generic Security Services (GSS) token for use on the iSeries. Enhanced Telnet clients will need to obtain the Kerberos services ticket from a Key Distribution Center (KDC). Implementation steps for acquiring the Kerberos services ticket will be limited to the Microsoft Security Support Provider Interface (SSPI) example below. For information on Kerberos services tickets, refer to your Network Authentication Service (NAS) documentation.

The custom USERVAR defined is:

USERVAR	VALUE	EXAMPLE	DESCRIPTION
IBMTICKET	binary(16384)	16384-byte hex field	Kerberos services token

Several other USERVARs, as defined in Section 6, can be used along with the IBMTICKET USERVAR to allow a user greater control over their startup environment.

The custom USERVARs defined to accomplish this are:

USERVAR	VALUE	EXAMPLE	DESCRIPTION
IBMCURLIB	us-ascii char(x)	QGPL	Current library
IBMIMENU	us-ascii char(x)	MAIN	Initial menu
IBMPROGRAM	us-ascii char(x)	QCMD	Program to call

x - up to a maximum of 10 characters

If you use the IBMSENDCONFREC USERVAR, as described in Section 5, with a value of YES along with the Kerberos ticket USERVARs described above, you will receive a Startup Response Record that will contain a response code informing your Telnet client of the success or failure of the Kerberos validation attempt. See Section 11 for details on the Startup Response Record.

The following Microsoft SSPI example illustrates how to get the client security token, which contains the Kerberos services ticket.

- 1) Get a handle to the user's credentials:

```

PSecurityFunctionTable pSSPI_;
CredHandle credHandle;
TimeStamp timeStamp;

ss = pSSPI_->AcquireCredentialsHandle(
    NULL,                      // Principal
    "Kerberos",                // PackageName
    SECPKG_CRED_OUTBOUND,      // CredentialUse
    NULL,                      // LogonID
    NULL,                      // AuthData
    NULL,                      // GetKeyFnc
    NULL,                      // GetKeyArg
    &credHandle,               // CredHandle
    &timeStamp);              // ExpireTime

```

- 2) Initialize security context to "request delegation". Mutual authentication is also requested, although it is not required and may not be performed.

```

Ctxthandle newContext;
unsigned long contextAttr;
unsigned char token[16384] ;
unsigned long tokenLen = sizeof(token);
SecBuffer sbo = {tokenLen, SECBUFFER_TOKEN, token};
SecBufferDesc sbdo = {SECBUFFER_VERSION, 1, &sbo}

pSSPI_->InitializeSecurityContext(
    &credHandle,                // CredHandle
    NULL,                      // Context
    "krbsrv400/fullyqualifiedLowerCaseSystemName",
                                // ServicePrincipalName
    ISC_REQ_CONNECTION|ISC_REQ_DELEGATE|ISC_REQ_MUTUAL_AUTH,
                                // ContextRequest
    NULL,                      // Reserved
    SECURITY_NATIVE_DREP,       // DataRep
    NULL,                      // Input
    NULL,                      // Reserved
    &newContext,                // NewContext
    &sbdo,                      // Output
    &contextAttr,               // ContextAttr
    &timeStamp);               // ExpireTime

```

- 3) Free the user credentials handle with FreeCredentialsHandle().
- 4) Send security token to Telnet Server (padded with escape characters).

The following illustrates the Kerberos Token Negotiation:

```

iSeries Telnet server          Enhanced Telnet client
-----
IAC DO NEW-ENVIRON             -->
                                <-- IAC WILL NEW-ENVIRON

IAC SB NEW-ENVIRON SEND
USERVAR "IBMRSEEDxxxxxxxx"
VAR USERVAR IAC SE             -->
                                IAC SB NEW-ENVIRON IS
                                USERVAR "IBMTICKET" VALUE
                                "zzzzzzzz..."
                                <-- IAC SE
                                .
                                .
                                .
(other negotiations)           .

```

In this example, "xxxxxxxx" is an 8-byte hexadecimal random server seed, and "zzzzzzzz..." is the complete Kerberos services token. If the Kerberos services token is not valid, then the sign-on panel is not bypassed. It should be noted that for the Kerberos token a random server seed is not needed, although it will be sent by the Telnet Server.

Actual bytes transmitted in the above example are shown in hex below, where the server seed is "7D3E488F18080404", and the Kerberos services token starts with "DFB0402F22ABA3BA...". The complete Kerberos services token is not shown here, as the length of the token could be 16384 bytes and would make this document extremely large. As described in Section 6, the client must escape and/or byte stuff any Kerberos token bytes, which could be seen as a Telnet environment option [RFC1572], specifically VAR, VALUE, ESC, and USERVAR.

```

iSeries Telnet server          Enhanced Telnet client
-----
FF FD 27                       -->
                                <-- FF FB 27

FF FA 27 01 03 49 42 4D
52 53 45 45 44 7D 3E 48
8F 18 08 04 04 00 03 FF
F0                               -->
                                FF FA 27 00 03 49 42 4D
                                54 49 43 48 45 54 01 DF
                                B0 40 2F 22 AB A3 BA...
                                <-- FF F0

```

## 7. Device Name Collision Processing

Device name collision occurs when a Telnet client sends the Telnet server a virtual device name that it wants to use, but that device is already in use on the server. When this occurs, the Telnet server sends a request to the client asking it to try another device name. The environment option negotiation uses the USERVAR name of DEVNAME to communicate the virtual device name. The following shows how the Telnet server will request the Telnet client to send a different DEVNAME when device name collision occurs.

iSeries Telnet server	Enhanced Telnet client
-----	-----
IAC SB NEW-ENVIRON SEND	
VAR USERVAR IAC SE	-->

Server requests all environment variables be sent.

	IAC SB NEW-ENVIRON IS USERVAR
	"DEVNAME" VALUE "MYDEVICE1"
	USERVAR "xxxxx" VALUE "xxx"
	...
<--	IAC SE

Client sends all environment variables, including DEVNAME. Server tries to select device MYDEVICE1. If the device is already in use, server requests DEVNAME be sent again.

IAC SB NEW-ENVIRON SEND	
USERVAR "DEVNAME" IAC SE	-->

Server sends a request for a single environment variable: DEVNAME

	IAC SB NEW-ENVIRON IS USERVAR
<--	"DEVNAME" VALUE "MYDEVICE2" IAC SE

Client sends one environment variable, calculating a new value of MYDEVICE2. If MYDEVICE2 is different from the last request, then server tries to select device MYDEVICE2, else server disconnects client. If MYDEVICE2 is also in use, server will send DEVNAME request again and keep doing so until it receives a device that is not in use, or the same device name twice in row.

## 8. Enhanced Printer Emulation Support

Telnet environment option USERVARs have been defined to allow a compliant Telnet client more control over the Telnet server virtual device on the iSeries. These USERVARs allow the client Telnet to select a previously created virtual device or auto-create a new virtual device with requested attributes.

This makes the enhancements available to any Telnet client that chooses to support the new negotiations.

The USERVARs defined to accomplish this are:

USERVAR	VALUE	EXAMPLE	DESCRIPTION
DEVNAME	us-ascii char(x)	PRINTER1	Printer device name
IBMIGCFEAT	us-ascii char(6)	2424J0	IGC feature (DBCS)
IBMMSGQNAME	us-ascii char(x)	QSYSOPR	*MSGQ name
IBMMSGQLIB	us-ascii char(x)	QSYS	*MSGQ library
IBMFONT	us-ascii char(x)	12	Font
IBMFORMFEED	us-ascii char(1)	C   U   A	Formfeed
IBMTRANSFORM	us-ascii char(1)	1   0	Transform
IBMFMRTYPMDL	us-ascii char(x)	*IBM42023	Mfg. type and model
IBMPPRSRC1	binary(1)	1-byte hex field	Paper source 1
IBMPPRSRC2	binary(1)	1-byte hex field	Paper source 2
IBMENVELOPE	binary(1)	1-byte hex field	Envelope hopper
IBMASCI899	us-ascii char(1)	1   0	ASCII 899 support
IBMWSCSTNAME	us-ascii char(x)	*NONE	WSCST name
IBMWSCSTLIB	us-ascii char(x)	*LIBL	WSCST library

x - up to a maximum of 10 characters

The "IBM" prefix on the USERVARs denotes iSeries-specific attributes.

The DEVNAME USERVAR is used for both displays and printers. The IBMFONT and IBMASCI899 are used only for SBCS environments.

For a description of most of these parameters (drop the "IBM" from the USERVAR) and their permissible values, refer to Chapter 8 in the Communications Configuration Reference [COMM-CONFIG].

The IBMIGCFEAT supports the following variable DBCS language identifiers in position 5 (positions 1-4 must be '2424'; position 6 must be '0'):

'J' = Japanese	'K' = Korean
'C' = Traditional Chinese	'S' = Simplified Chinese

The IBMTRANSFORM and IBMASCII899 values correspond to:

'1' = Yes    '0' = No

The IBMFORMFEED values correspond to:

'C' = Continuous    'U' = Cut    'A' = Autocut

The IBMPPRSRC1, IBMPPRSRC2, and IBMENVELOPE custom USERVARs do not map directly to their descriptions in Chapter 8 in the Communications Configuration Reference [COMM-CONFIG]. To map these, use the index listed here:

IBMPPRSRC1	HEX	IBMPPRSRC2	HEX	IBMENVELOPE	HEX
-----	----	-----	----	-----	----
*NONE	'FF'X	*NONE	'FF'X	*NONE	'FF'X
*MFRTYPMDL	'00'X	*MFRTYPMDL	'00'X	*MFRTYPMDL	'00'X
*LETTER	'01'X	*LETTER	'01'X	*B5	'06'X
*LEGAL	'02'X	*LEGAL	'02'X	*MONARCH	'09'X
*EXECUTIVE	'03'X	*EXECUTIVE	'03'X	*NUMBER9	'0A'X
*A4	'04'X	*A4	'04'X	*NUMBER10	'0B'X
*A5	'05'X	*A5	'05'X	*C5	'0C'X
*B5	'06'X	*B5	'06'X	*DL	'0D'X
*CONT80	'07'X	*CONT80	'07'X		
*CONT132	'08'X	*CONT132	'08'X		
*A3	'0E'X	*A3	'0E'X		
*B4	'0F'X	*B4	'0F'X		
*LEDGER	'10'X	*LEDGER	'10'X		

## 9. Telnet Printer Terminal Types

New Telnet options are defined for the printer pass-through mode of operation. To enable printer pass-through mode, both the client and server must agree to support at least the Transmit-Binary, End-Of-Record, and Terminal-Type Telnet options. The following are new terminal types for printers:

TERMINAL-TYPE	DESCRIPTION
-----	-----
IBM-5553-B01	Double-Byte printer
IBM-3812-1	Single-Byte printer

Specific characteristics of the IBM-5553-B01 or IBM-3812-1 printers are specified through the USERVAR IBMMFRTYPMDL, which specifies the manufacturer type and model.

An example of a typical negotiation process to establish printer pass-through mode of operation is shown below. In this example, the server initiates the negotiation by sending the DO TERMINAL-TYPE request.

For DBCS environments, if IBMTRANSFORM is set to 1 (use Host Print Transform), then the virtual device created is 3812, not 5553. Therefore, IBM-3812-1 (and not IBM-5553-B01) should be negotiated for TERMINAL-TYPE.

iSeries Telnet server		Enhanced Telnet client
-----		-----
IAC DO NEW-ENVIRON	-->	
	<--	IAC WILL NEW-ENVIRON
IAC SB NEW-ENVIRON SEND		
VAR USERVAR IAC SE	-->	IAC SB NEW-ENVIRON IS USERVAR "DEVNAME" VALUE "PCPRINTER" USERVAR "IBMMSGQNAME" VALUE "QSYSOPR" USERVAR "IBMMSGQLIB" VALUE "*LIBL" USERVAR "IBMTRANSFORM" VALUE "0" USERVAR "IBMFONT" VALUE "12" USERVAR "IBMFORMFEED" VALUE "C" USERVAR "IBMPPRSRC1" VALUE ESC '01'X USERVAR "IBMPPRSRC2" VALUE '04'X USERVAR "IBMENVELOPE" VALUE IAC 'FF'X
	<--	IAC SE
IAC DO TERMINAL-TYPE	-->	
	<--	IAC WILL TERMINAL-TYPE
IAC SB TERMINAL-TYPE SEND		
IAC SE	-->	IAC SB TERMINAL-TYPE IS IBM-3812-1
	<--	IAC SE
IAC DO BINARY	-->	
	<--	IAC WILL BINARY
IAC DO EOR	-->	
	<--	IAC WILL EOR

Some points about the above example. The IBMPPRSRC1 value requires escaping the value using ESC according to Telnet environment options [RFC1572]. The IBMPPRSRC2 does not require an ESC character since '04'X has no conflict with environment options. Finally, to send 'FF'X for the IBMENVELOPE value, escape the 'FF'X value by using another 'FF'X (called "doubling"), so as not to have the value interpreted as a Telnet character per the Telnet protocol specification [RFC854].



Actual bytes transmitted in the above example are shown in hex below.

iSeries Telnet server		Enhanced Telnet client
-----		-----
FF FD 27	-->	
	<--	FF FB 27
FF FA 27 01 00 03 FF F0	-->	
		FF FA 27 00 03 44 45 56
		4E 41 4D 45 01 50 43 50
		52 49 4E 54 45 52 03 49
		42 4D 4D 53 47 51 4E 41
		4D 45 01 51 53 59 53 4F
		50 52 03 49 42 4D 4D 53
		47 51 4C 49 42 01 2A 4C
		49 42 4C 03 49 42 4D 54
		52 41 4E 53 46 4F 52 4D
		01 30 03 49 42 4D 46 4F
		4E 54 01 31 32 03 49 42
		4D 46 4F 52 4D 46 45 45
		44 01 43 03 49 42 4D 50
		50 52 53 52 43 31 01 02
		01 03 49 42 4D 50 50 52
		53 52 43 32 01 04 03 49
		42 4D 45 4E 56 45 4C 4F
	<--	50 45 01 FF FF FF F0
FF FD 18	-->	
	<--	FF FB 18
FF FA 18 01 FF F0	-->	
		FF FA 18 00 49 42 4D 2D
	<--	33 38 31 32 2D 31 FF F0
FF FD 00	-->	
	<--	FF FB 00
FF FD 19	-->	
		FF FB 19

#### 10. Startup Response Record for Printer and Display Devices

Once Telnet negotiation for a 5250 pass-through mode is completed, the iSeries Telnet server will initiate a virtual device (printer or display) power-on sequence on behalf of the Telnet client. The Telnet server will supply a Startup Response Record to the Telnet client with the status of the device power-on sequence, indicating success or failure of the virtual device power-on sequence.

This section shows an example of two Startup Response Records. The source device is a type 3812 model 01 printer with the name "PCPRINTER" on the target system "TARGET".

Figure 1 shows an example of a successful response; Figure 2 shows an example of an error response.

### 10.1. Example of a Success Response Record

The response record in Figure 1 was sent by an iSeries at Release V4R2. It is an example of the target sending back a successful Startup Response Record.

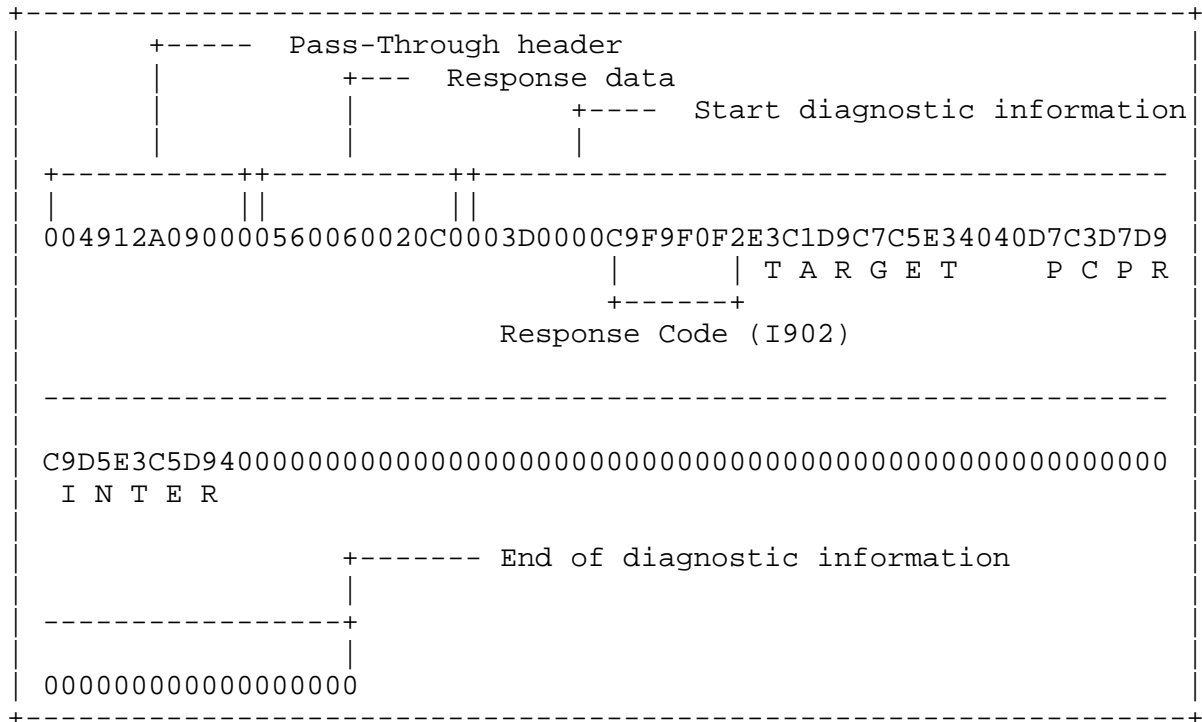


Figure 1. Example of a success response record

- ```
- '0049'X = Length pass-through data, including this length field
- '12A0'X = GDS LU6.2 header
- '90000560060020C0003D0000'X = Fixed value fields
- 'C9F9F0F2'X = Response Code (I902)
- 'E3C1D9C7C5E34040'X = System Name (TARGET)
- 'D7C3D7D9C9D5E3C5D940'X = Object Name (PCPRINTER)
```

## 10.2. Example of an Error Response Record

The response record in Figure 2 is one that reports an error. The virtual device named "PCPRINTER" is not available on the target system "TARGET" because the device is not available. You would normally see this error if the printer were already assigned to another Telnet session.

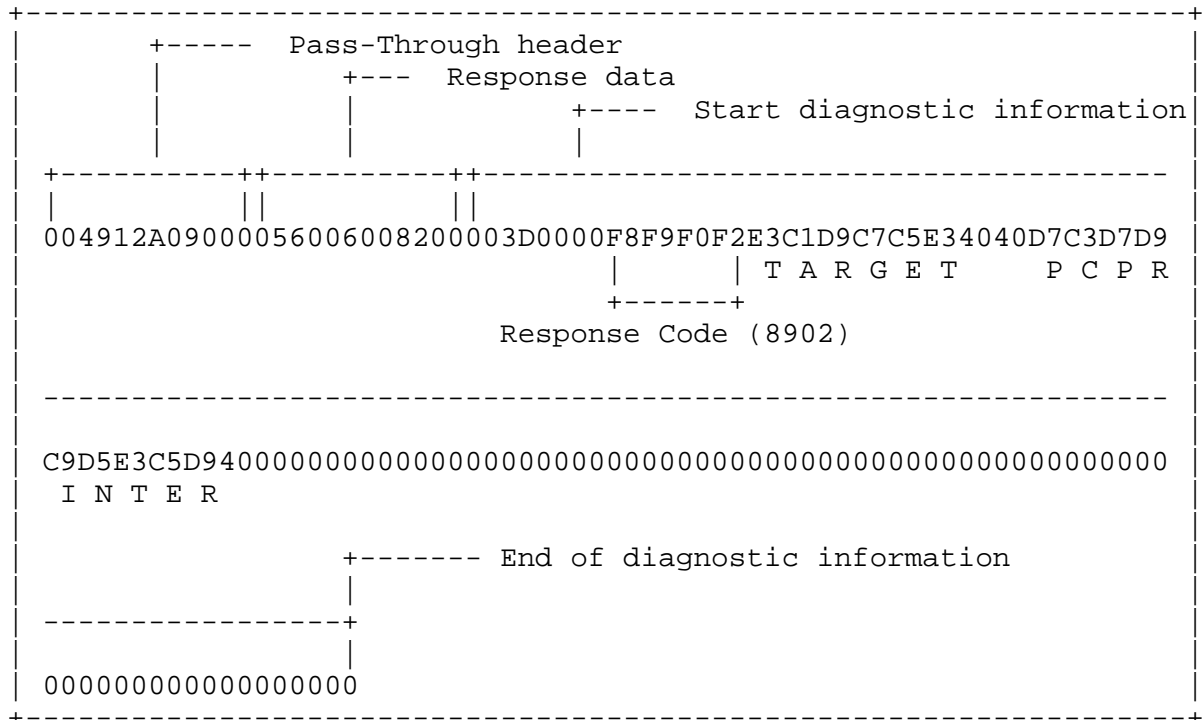


Figure 2. Example of an error response record

- ```
- '0049'X = Length pass-through data, including this length field
- '12A0'X = GDS LU6.2 header
- '90000560060020C0003D0000'X = Fixed value fields
- 'F8F9F0F2'X = Response Code (8902)
- 'E3C1D9C7C5E34040'X = System Name (TARGET)
- 'D7C3D7D9C9D5E3C5D940'X = Object Name (PCPRINTER)
```

### 10.3. Example of a Response Record with Device Name Retry

The Response Record can be used in conjunction with the DEVNAME Environment variable to allow client emulators to inform users of connection failures. In addition, this combination could be used by client emulators that accept multiple device names to try on session connections. The client would be able to walk through a list of possible device names and provide feedback based on the response code(s) received for each device name that was rejected.

The following sequence shows a negotiation between the client and the server in which a named device "RFCTEST" is requested by the client. The device name is already assigned to an existing condition. The server responds with the Response Record showing an 8902 response code. The client could use this information to inform the user that the device name just tried was already in use. Following the Response Record the server would then invite the client to try another device name. Because the same device name was used again by the client, the server closed the session.

iSeries Telnet server		Enhanced Telnet client
-----		-----
IAC DO NEW-ENVIRON	-->	
	<--	IAC WILL NEW-ENVIRON
IAC DO TERMINAL-TYPE	-->	
	<--	IAC WILL TERMINAL-TYPE
IAC SB NEW-ENVIRON SEND USERVAR "IBMRSEEDxxxxxxxxx" VAR USERVAR IAC SE	-->	
		IAC SB NEW-ENVIRON IS USERVAR "DEVNAME" VALUE "RFCTEST" USERVAR "IBMSENDCONFREC" VALUE "YES"
	<--	IAC SE
IAC SB TERMINAL-TYPE SEND IAC SE	-->	
	<--	IAC SB TERMINAL-TYPE IS IBM-3180-2 IAC SE (terminal type negotiations completed)
IAC DO EOR	-->	
	<--	IAC WILL EOR
IAC WILL EOR	-->	
	<--	IAC DO EOR
IAC DO BINARY	-->	
	<--	IAC WILL BINARY
IAC WILL BINARY	-->	
	<--	IAC DO BINARY
(73 BYTE RFC 1205 RECORD WITH 8902 ERROR CODE)	-->	
IAC SB NEW-ENVIRON SEND USERVAR "DEVNAME" IAC SE	-->	
		IAC SB NEW-ENVIRON IS USERVAR "DEVNAME" VALUE "RFCTEST" USERVAR "IBMSENDCONFREC" VALUE "YES"
	<--	IAC SE
(server closes connection)		

Actual bytes transmitted in the above example are shown in hex below.

iSeries Telnet server		Enhanced Telnet client
-----		-----
FF FD 27	-->	
	<--	FF FB 27
FF FD 18	-->	
	<--	FF FB 18
FF FA 27 01 03 49 42 4D		
52 53 45 45 44 C4 96 67		
76 9A 23 E3 34 00 03 FF		
F0	-->	
		FF FA 27 00 03 44 45 56
		4E 41 4D 45 01 52 46 43
		54 45 53 54 03 49 42 4D
		53 45 4E 44 43 4F 4E 46
		52 45 43 01 59 45 53 FF
	<--	F0
FF FA 18 01 FF F0	-->	
	<--	FF FA 18 00 49 42 4D 2D
		33 31 38 30 2D 32 FF F0
FF FD 19	-->	
	<--	FF FB 19
FF FB 19	-->	
	<--	FF FD 19
FF FD 00	-->	
	<--	FF FB 00
FF FB 00	-->	
	<--	FF FD 00
00 49 12 A0 90 00 05 60		
06 00 20 C0 00 3D 00 00		
F8 F9 F0 F2 D9 E2 F0 F3		
F5 40 40 40 00 00 00 00		
00 00 00 00 00 00 00 00		
00 00 00 00 00 00 00 00		
00 00 00 00 00 00 00 00		
00 00 00 00 00 00 00 00		
00 00 00 00 00 00 00 00		
00 FF EF	-->	
FF FA 27 01 03 44 45 56		
4E 41 4D 45 FF F0	-->	
	<--	FF FA 27 00 03 44 45 56
		4E 41 4D 45 01 52 46 43
		54 45 53 54 03 49 42 4D
		53 45 4E 44 43 4F 4E 46
		52 45 43 01 59 45 53 FF
		F0

#### 10.4. Response Codes

The Start-Up Response Record success response codes:

CODE	DESCRIPTION
-----	-----
I901	Virtual device has less function than source device.
I902	Session successfully started.
I906	Automatic sign-on requested, but not allowed. Session still allowed; a sign-on screen will be coming.

The Start-Up Response Record error response codes:

CODE	DESCRIPTION
-----	-----
2702	Device description not found.
2703	Controller description not found.
2777	Damaged device description.
8901	Device not varied on.
8902	Device not available.
8903	Device not valid for session.
8906	Session initiation failed.
8907	Session failure.
8910	Controller not valid for session.
8916	No matching device found.
8917	Not authorized to object.
8918	Job canceled.
8920	Object partially damaged.
8921	Communications error.
8922	Negative response received.
8923	Start-up record built incorrectly.
8925	Creation of device failed.
8928	Change of device failed.
8929	Vary on or vary off failed.
8930	Message queue does not exist.
8934	Start-up for S/36 WSF received.
8935	Session rejected.
8936	Security failure on session attempt.
8937	Automatic sign-on rejected.
8940	Automatic configuration failed or not allowed.
I904	Source system at incompatible release.

The Start-Up Response Record error response codes for non-Kerberos Services Token automatic sign-on:

CODE	DESCRIPTION
----	-----
0001	System error.
0002	Userid unknown.
0003	Userid disabled.
0004	Invalid password/passphrase/token.
0005	Password/passphrase/token is expired.
0006	Pre-V2R2 password.
0008	Next invalid password/passphrase/token will revoke userid.

The Start-Up Response Record error response codes for Kerberos Services Token automatic sign-on support:

CODE	DESCRIPTION
----	-----
0001	User profile is disabled.
0002	Kerberos principal maps to a system user profile.
0003	Enterprise Identity Map (EIM) configuration error.
0004	EIM does not map Kerberos principal to user profile.
0005	EIM maps Kerberos principal to multiple user profiles.
0006	EIM maps Kerberos principal to user profile not found on system.
1000	None of the requested mechanisms are supported by the local system.
2000	The input name is not formatted properly or is not valid.
6000	The received input token contains an incorrect signature.
7000	No credentials available or credentials valid for context init only.
9000	Consistency checks performed on the input token failed.
A000	Consistency checks on the cred structure failed.
B000	Credentials are no longer valid.
D000	The runtime failed for reasons that are not defined at the GSS level.

In the case where the USERVAR, DEVNAME USERVAR, IBMSENDCONFREC USERVAR, IBMSUBSPW USERVAR, and IBMRSEED USERVAR are all used together, any device errors will take precedence over automatic sign-on errors. That is:

- 1) If the requested named device is not available or an error occurs when attempting to create the device on the server system, a device related return code (i.e., 8902) will be sent to the client system in the display confirmation record.



- 2) If the requested named device is available or no errors occur when attempting to create the device on the server system, an automatic sign-on return code (i.e., 0002) will be sent to the client system in the display confirmation record.

## 11. Printer Steady-State Pass-Through Interface

The information in this section applies to the pass-through session after the receipt of startup confirmation records is complete.

Following is the printer header interface used by Telnet.

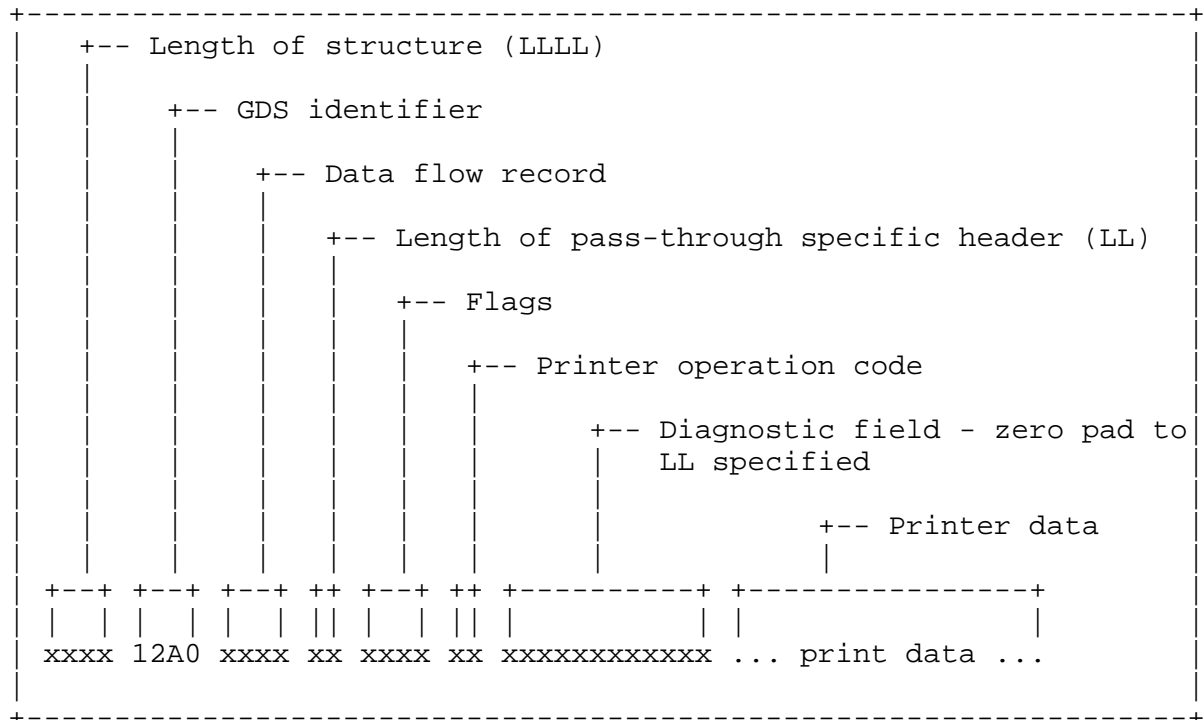


Figure 3. Layout of the printer pass-through header

BYTES 0-1: Length of structure including this field (LLLL)

BYTES 2-3: GDS Identifier ('12A0'X)

BYTE 4-5: Data flow record

This field contains flags that describe what type of data pass-through should be expected to be found following this header. Generally, bits 0-2 in the first byte are mutually exclusive (that is, if one of them is

set to '1'B, the rest will be set to '0'B.) The bits and their meanings follow.

BIT	DESCRIPTION
0	Start-Up confirmation
1	Termination record
2	Start-Up Record
3	Diagnostic information included
4 - 5	Reserved
6	Reserved
7	Printer record
8 - 13	Reserved
14	Client-originated (inbound) printer record
15	Server-originated (outbound) printer record

BYTE 6: Length printer pass-through header including this field (LL)

BYTES 7-8: Flags

BYTE 7 BITS: xxxx x111 --> Reserved  
               xxxx 1xxx --> Last of chain  
               xxx1 xxxx --> First of chain  
               xx1x xxxx --> Printer now ready  
               xlxx xxxx --> Intervention Required  
               1xxx xxxx --> Error Indicator

BYTE 8 BITS: xxxx xxxx --> Reserved

BYTE 9: Printer operation code

'01'X Print/Print complete  
 '02'X Clear Print Buffers

BYTE 10-LL: Diagnostic information (Note 1)

If BYTE 7 = xx1x xxxx, then bytes 10-LL may contain:  
     Printer ready                                   C9 00 00 00 02

If BYTE 7 = xlxx xxxx, then bytes 10-LL may contain: (Note 2)  
     Command/parameter not valid   C9 00 03 02 2x  
     Print check                    C9 00 03 02 3x  
     Forms check                    C9 00 03 02 4x  
     Normal periodic condition      C9 00 03 02 5x  
     Data stream error              C9 00 03 02 6x  
     Machine/print/ribbon check    C9 00 03 02 8x

If BYTE 7 = lxxx xxxx, then bytes 10-LL may contain: (Note 3)

Cancel	08 11 02 00
Invalid print parameter	08 11 02 29
Invalid print command	08 11 02 28

Diagnostic information notes:

1. LL is the length of the structure defined in Byte 6. If no additional data is present, the remainder of the structure must be padded with zeroes.
2. These are printer SIGNAL commands. Further information on these commands may be obtained from the 5494 Remote Control Unit Functions Reference guide [5494-CU]. Refer to your iSeries printer documentation for more specific information on these data stream exceptions. The following are some 3812 and 5553 errors that may be seen:

Machine check	C9 00 03 02 11
Graphics check	C9 00 03 02 26
Print check	C9 00 03 02 31
Form jam	C9 00 03 02 41
Paper jam	C9 00 03 02 47
End of forms	C9 00 03 02 50
Printer not ready	C9 00 03 02 51
Data stream - class 1	C9 00 03 02 66 loss of text
Data stream - class 2	C9 00 03 02 67 text appearance
Data stream - class 3	C9 00 03 02 68 multibyte control error
Data stream - class 4	C9 00 03 02 69 multibyte control parm
Cover unexpectedly open	C9 00 03 02 81
Machine check	C9 00 03 02 86
Machine check	C9 00 03 02 87
Ribbon check	C9 00 03 02 88

3. These are printer negative responses. Further information on these commands may be obtained from the 5494 Remote Control Unit Functions Reference guide [5494-CU].

The print data will start in byte LL+1.

#### 11.1. Example of a Print Record

Figure 4 shows the server sending the client data with a print record. This is normally seen following receipt of a Success Response Record, such as the example in Figure 1.

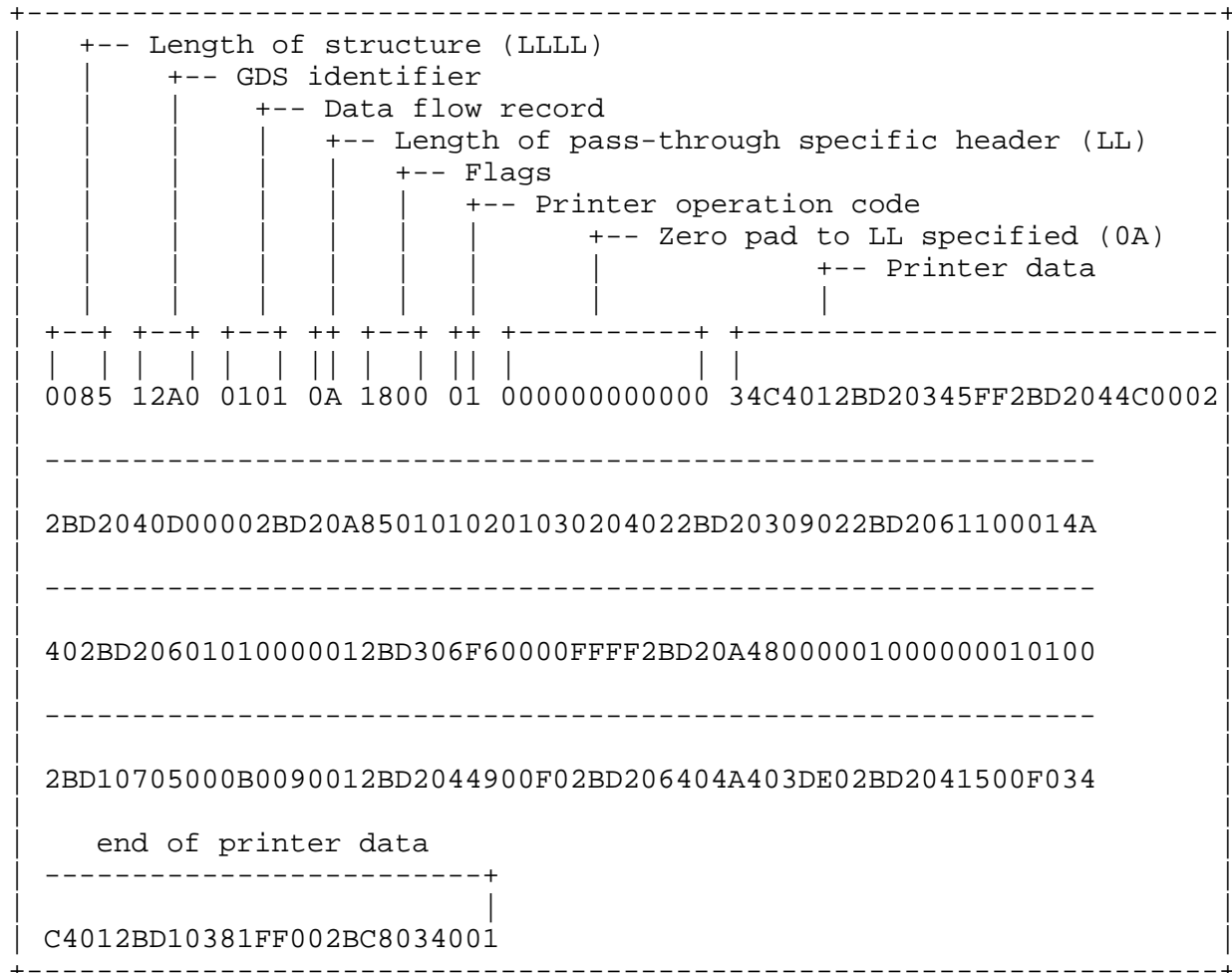


Figure 4. Server sending client data with a print record

- '0085'X = Logical record length, including this byte (LLLL)
- '12A0'X = GDS LU6.2 header
- '0101'X = Data flow record (server to client)
- '0A'X = Length of pass-through specific header (LL)
- '1800'X = First of chain / Last of chain indicators
- '01'X = Print
- '000000000000'X = Zero pad header to LL specified
- '34C401'X = First piece of data for spooled data
- Remainder is printer data/commands/orders

### 11.2. Example of a Print Complete Record

Figure 5 shows the client sending the server a print complete record. This would normally follow receipt of a print record, such as the example in Figure 4. This indicates successful completion of a print request.

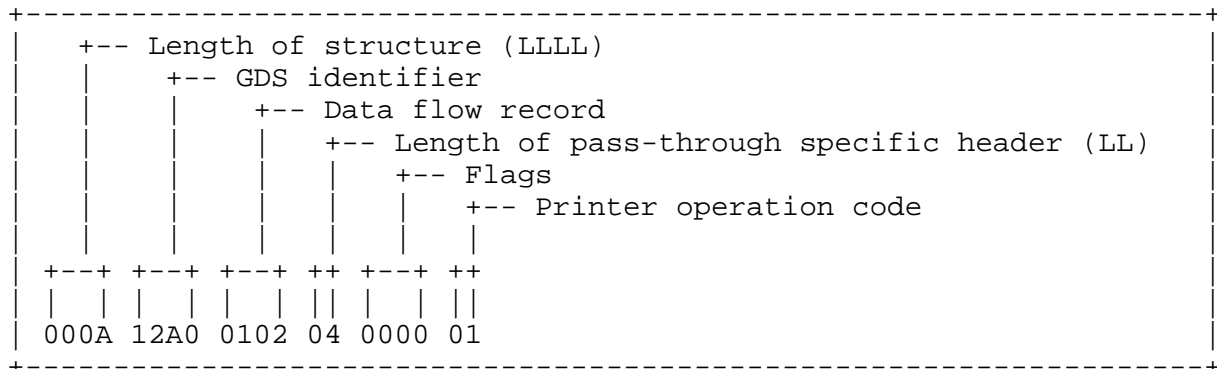


Figure 5. Client sending server a print complete record

- ```
- '000A'X = Logical record length, including this byte (LLLL)
- '12A0'X = GDS LU6.2 header
- '0102'X = Data flow response record (client to server)
- '04'X    = Length of pass-through specific header (LL)
- '0000'X = Good Response
- '01'X    = Print Complete
```

### 11.3. Example of a Null Print Record

Figure 6 shows the server sending the client a null print record. The null print record is the last print command the server sends to the client for a print job, and it indicates to the printer that there is no more data. The null data byte '00'X is optional and in some cases may be omitted (in particular, this scenario occurs in DBCS print streams).

This example would normally follow any number of print records, such as the example in Figure 4. This indicates successful completion of a print job. The client normally responds to this null print record with another print complete record, such as in Figure 5.

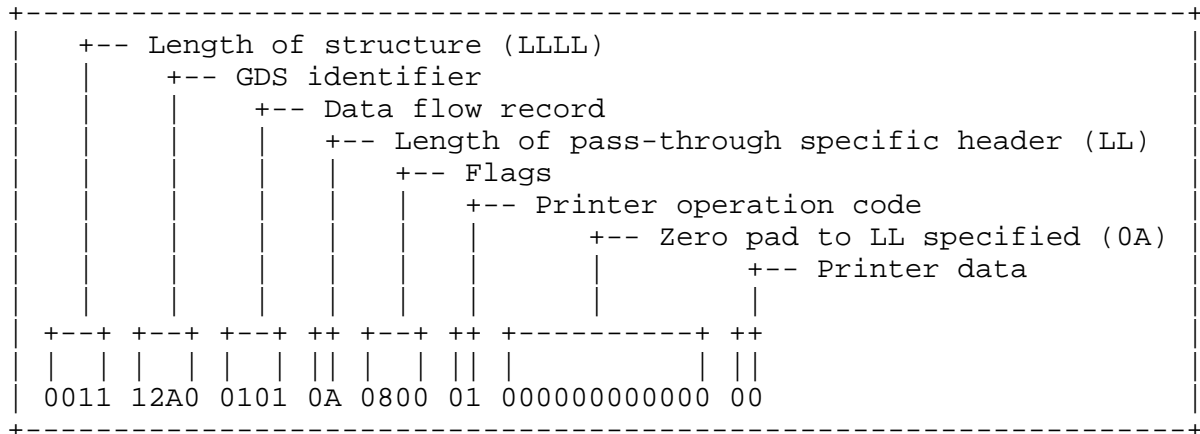


Figure 6. Server sending client a null print record

- ```
- '0011'X      = Logical record length, including this byte
- '12A0'X      = GDS LU6.2 header
- '0101'X      = Data flow record
- '0A'X        = Length of pass-through specific header (LL)
- '0800'X      = Last of Chain
- '01'X        = Print
- '000000000000'X = Zero pad header to LL specified
- '00'X        = Null data byte
```

## 12. End-to-End Print Example

The next example shows a full print exchange between a Telnet client and server for a 526 byte spooled file. Selective translation of the hexadecimal streams into 1) Telnet negotiations and 2) ASCII/EBCDIC characters is done to aid readability. Telnet negotiations are delimited by '(' and ')' parenthesis characters; ASCII/EBCDIC conversions are bracketed by '|' vertical bar characters.

```

iSeries Telnet server          Enhanced Telnet client
-----
FFFD27                          -->

(IAC DO NEW-ENVIRON)

                                <-- FFFB27

                                (IAC WILL NEW-ENVIRON)

FFFD18FFFA270103 49424D5253454544
7EA5DFDDFD300404 0003FFF0      -->

(IAC DO TERMINAL-TYPE
IAC SB NEW-ENVIRON SEND USERVAR
IBMRSEED xxxxxxxx VAR USERVAR
IAC SE)

                                <-- FFFB18

                                (IAC WILL TERMINAL-TYPE)

FFFA1801FFF0                    -->

(IAC SB TERMINAL-TYPE SEND IAC
SE)

                                FFFA27000349424D 52534545447EA5DF
                                DDFD300404000344 45564E414D450144
                                554D4D5950525403 49424D4D5347514E
                                414D450151535953 4F50520349424D4D
                                5347514C4942012A 4C49424C0349424D
                                464F4E5401313103 49424D5452414E53
                                464F524D01310349 424D4D4652545950
                                4D444C012A485049 490349424D505052
                                5352433101020103 49424D5050525352
                                433201040349424D 454E56454C4F5045
                                01FFFFF0349424D41 5343494938393901
                                <-- 30FFF0

```

```

(IAC SB NEW-ENVIRON IS USERVAR
  IBMRSEED xxxxxxxx VAR
  USERVAR DEVNAME VALUE DUMMYPR
  USERVAR IBMMSGQNAME VALUE
  QSYSOPR
  USERVAR IBMMSGQLIB VALUE *LIBL
  USERVAR IBMFONT VALUE 11
  USERVAR IBMTRANSFORM VALUE 1
  USERVAR IBMMFRTYPMDL VALUE *HPII
  USERVAR IBMPPRSRC1 VALUE
  ESC '01'X
  USERVAR IBMPPRSRC2 VALUE '04'X
  USERVAR IBMENVELOPE VALUE IAC
  USERVAR IBMASCII899 VALUE 0
  IAC SE)

<-- FFFA180049424D2D 333831322D31FFF0

(IAC SB TERMINAL-TYPE IS
  IBM-3812-1 IAC SE)
FFFD19 -->

(IAC DO EOR)

<-- FFFB19

(IAC WILL EOR)

FFFB19 -->

(IAC WILL EOR)

<-- FFFD19

(IAC DO EOR)
FFFD00 -->

(IAC DO BINARY)

<-- FFFB00

(IAC WILL BINARY)
FFFB00 -->

(IAC WILL BINARY)

<-- FFFD00

(IAC DO BINARY)

```



004912A090000560	060020C0003D0000		- {		
C9F9F0F2C5D3C3D9	E3D7F0F6C4E4D4D4		I902ELCRTP06DUMM		(EBCDIC)
E8D7D9E340400000	0000000000000000		YPRT		
0000000000000000	0000000000000000				
0000000000000000	00FFEF				

--&gt;

(73-byte startup success response  
record ... IAC EOR)

00DF12A001010A18	0001000000000000				
03CD1B451B283130	551B287330703130		E (10U (s0p10		(ASCII)
2E30306831327630	733062303033541B		.00h12v0s0b003T		
287330421B266440	1B266C304F1B266C		(s0B &d@ &l0O &l		
303038431B266C30	3035431B28733070		008C &l005C (s0p		
31372E3130683130	7630733062303030		17.10h10v0s0b000		
541B283130551B28	73307031372E3130		T (10U (s0p17.10		
6831307630733062	303030541B287330		h10v0s0b000T (s0		
421B2664401B266C	314F1B266C303035		B &d@ &l1O &l005		
431B287330703137	2E31306831307630		C (s0p17.10h10v0		
733062303030541B	266C314F1B287330		s0b000T &l1O (s0		
7031372E31306831	3076307330623030		p17.10h10v0s0b00		
30541B2873307031	372E313068313076		0T (s0p17.10h10v		
3073306230303054	1B266C30303543FF		0s0b000T &l005C		

EF

--&gt;

(... 223-byte print record ...  
... first of chain ...  
... last of chain ... IAC EOR)

&lt;-- 000A12A001020400 0001FFEF

031012A001010A10	0001000000000000		(10-byte print complete header)		
03FFFF1B451B2831	30551B2873307031		E (10U (s0p1		(ASCII)
372E313068313076	3073306230303054		7.10h10v0s0b000T		
1B287330421B2664	401B266C314F1B26		(s0B &d@ &l1O &		
6C303035431B266C	31481B266C314F1B		1005C &l1H &l1O		
266C3032411B266C	31431B266C303030		&l02A &l1C &l000		
38451B266C303038	431B266C30303439		8E &l008C &l0049		
461B266130521B26	6C303035430A0A0A		F &a0R &l005C		
0A0A0A0A1B26612B	3030303130561B26		&a+00010V &		
6C303035431B2661	2B30303231364820		1005C &a+00216H		
2020202020202020	2020202020202020				
2020202020205072	696E74204B657920		Print Key		
4F75747075742020	2020202020202020		Output		
2020202020202020	2020202020202020				
2020202020205061	6765202020310D0A		Page 1		
1B26612B30303231	3648202020203537		&a+00216H 57		
3639535331205634	52334D3020393830		69SS1 V4R3M0 980		
373203FFFF392020	2020202020202020		72 9		

202020202020454C	4352545030362020	ELCRTP06
2020202020202020	202030332F33312F	03/31/
3939202031363A33	303A34350D0A1B26	99 16:30:45 &
612B303032313648	0D0A1B26612B3030	a+00216H &a+00
3231364820202020	446973706C617920	216H Display
4465766963652020	2E202E202E202E20	Device . . . .
2E203A2020515041	444556303033510D	. : QPADEV003Q
0A1B26612B303032	3136482020202055	&a+00216H U
736572202020202E	202E202E202E202E	ser . . . . .
202E202E202E202E	203A202052434153	. . . . : RCAS
54524F0D0A1B2661	2B3030323136480D	TRO &a+00216H
0A1B26612B303032	313648204D41494E	&a+00216H MAIN
2020202020202020	2020202020202020	
2020202020202020	20202041532F3430	AS/40
30204D61696E204D	656E750D0A1B2661	0 Main Menu &a
2B30303203FFFF31	3648202020202020	+002 16H
2020202020202020	2020202020202020	
2020202020202020	2020202020202020	
2020202020202020	2020202020202020	
2020202020202053	797374656D3A2020	System:
20454C4352545030	360D0A1B26612B30	ELCRTP06 &a+0
3032313648205365	6C656374206F6E65	0216H Select one
206F662074686520	666F6C6C6F77696E	of the followin
673A0D0A1B26612B	3030323136480D0A	g: &a+00216H
1B26612B30303231	3648202020202020	&a+00216H
312E205573657220	7461736B730D0A1B	1. User tasks
26612B3030323136	4820202020202032	&a+00216H 2
2E204F6666696365	207461736B730D0A	. Office tasks
1B26612B30303231	36480D0A1B26612B	&a+00216H &a+
3030323136482020	20202020342E2046	00216H 4. F
696C65732C206C69	627261726965732C	iles, libraries,
20616EFFFF		an

(... 784-byte print record ...  
... first of chain ... IAC EOR)

<-- 000A12A001020400 0001FFEF

(10-byte print complete header)

020312A001010A00	0001000000000000		
64206603FFFF6F6C	646572730D0A1B26	d f olders &	(ASCII)
612B303032313648	0D0A1B26612B3030	a+00216H &a+00	
3231364820202020	2020362E20436F6D	216H 6. Com	
6D756E6963617469	6F6E730D0A1B2661	munications &a	
2B3030323136480D	0A1B26612B303032	+00216H &a+002	
3136482020202020	20382E2050726F62	16H 8. Prob	
6C656D2068616E64	6C696E670D0A1B26	lem handling &	
612B303032313648	202020202020392E	a+00216H 9.	

20446973706C6179	2061206D656E750D	Display a menu	
0A1B26612B303032	3136482020202020	&a+00216H	
31302E20496E666F	726D6174696F6E20	10. Information	
417373697374616E	74206F7074696F6E	Assistant option	
730D0A1B26612B30	3032313648202020	s &a+00216H	
202031312E20436C	69656E7420416363	11. Client Acc	
6573732F34303020	7461736B730D0A1B	ess/400 tasks	
26612B3030323136	480D0A1B26612B30	&a+00216H &a+0	
303231364803ED20	2020202039302E20	0216H 90.	
5369676E206F6666	0D0A1B26612B3030	Sign off &a+00	
323136480D0A1B26	612B303032313648	216H &a+00216H	
2053656C65637469	6F6E206F7220636F	Selection or co	
6D6D616E640D0A1B	26612B3030323136	mmand &a+00216	
48203D3D3D3E0D0A	1B26612B30303231	H ==> &a+0021	
36480D0A1B26612B	3030323136482046	6H &a+00216H F	
333D457869742020	2046343D50726F6D	3=Exit F4=Prom	
707420202046393D	5265747269657665	pt F9=Retrieve	
2020204631323D43	616E63656C202020	F12=Cancel	
4631333D496E666F	726D6174696F6E20	F13=Information	
417373697374616E	740D0A1B26612B30	Assistant &a+0	
3032313648204632	333D53657420696E	0216H F23=Set in	
697469616C206D65	6E750D0A1B26612B	itial menu &a+	
3030323136480D0A	1B26612B30303231	00216H &a+0021	
36480D0CFFEF		6H	

(... 515-byte print record ...  
IAC EOR)

<-- 000A12A001020400 0001FFEF

001412A001010A00 0001000000000000  
03021B45FFEF

(10-byte print complete header)  
| E | (ASCII)

(... 20-byte print record ...  
IAC EOR)

<-- 000A12A001020400 0001FFEF

001112A001010A08 0001000000000000  
00FFEF

(10-byte print complete header)

-->

(... 17-byte NULL print record ...  
... last of chain ... IAC EOR)

<-- 000A12A001020400 0001FFEF

(10-byte print complete header)

### 13. Security Considerations

The auto-sign-on feature provided by this RFC describes a way to encrypt your login password. However, while passwords can now be encrypted by using the IBMRSEED and IBMSUBSPW USERVAR negotiations, users should understand that only the login passwords are encrypted and not the entire Telnet session. Encryption of the Telnet session requires that another protocol layer, such as SSL, be added.

The auto-sign-on feature supports plain text passwords, encrypted passwords, and Kerberos tokens. However, using plain text passwords is strongly discouraged. iSeries system administrators may want to configure their systems to reject plain text passwords.

### 14. IANA Considerations

IANA registered the terminal types "IBM-3812-1" and "IBM-5553-B01" as a terminal type [RFC1091]. They are used when communicating with iSeries Telnet servers.

### 15. Normative References

- [RFC854] Postel, J. and J. Reynolds, "Telnet Protocol Specification", STD 8, RFC 854, May 1983.
- [RFC855] Postel, J. and J. Reynolds, "Telnet Option Specifications", STD 8, RFC 855, May 1983.
- [RFC1091] VanBokkelen, J., "Telnet terminal-type option", RFC 1091, February 1989.
- [RFC1205] Chmielewski, P., "5250 Telnet Interface", RFC 1205, February 1991.
- [RFC1572] Alexander, S., "Telnet Environment Option", RFC 1572, January 1994.
- [RFC2877] Murphy, T., Jr., Rieth, P., and J. Stevens, "5250 Telnet Enhancements", RFC 2877, July 2000.

### 16. Informative References

- [RFC856] Postel, J. and J. Reynolds, "Telnet Binary Transmission", STD 27, RFC 856, May 1983.
- [RFC858] Postel, J. and J. Reynolds, "Telnet Suppress Go Ahead Option", STD 29, RFC 858, May 1983.

- [RFC885] Postel, J., "Telnet end of record option", RFC 885, December 1983.
- [5494-CU] IBM, "5494 Remote Control Unit, Functions Reference", SC30-3533-04, August 1995.
- [SYSTEM-API] IBM, "AS/400 System API Reference", SC41-5801-01, February 1998.
- [COMM-CONFIG] IBM, "AS/400 Communications Configuration", SC41-5401-00, August 1997.
- [NLS-SUPPORT] IBM, "AS/400 National Language Support", SC41-5101-01, February 1998.
- [FIPS-46-2] Data Encryption Standard (DES), Federal Information Processing Standards Publication 46-2, January 22, 1988.
- [FIPS-81] DES Modes of Operation, Federal Information Processing Standards Publication 81, December 1980.
- [FIPS-180-1] Secure Hash Standard, Federal Information Processing Standards Publication 180-1, May 11, 1993.

#### 17. Relation to Other RFCs

This RFC relies on the 5250 Telnet Interface [RFC1205] in all examples.

This RFC replaces 5250 Telnet Enhancements [RFC2877], adding new sections for Kerberos, SHA-1, security and IANA considerations. Minor corrections and additional examples were also added.

Informative references have been removed.

## Authors' Addresses

Thomas E. Murphy, Jr.  
IBM Corporation  
2455 South Road  
Poughkeepsie, NY 12601

Phone: (845) 435-7063  
Fax: (845) 432-9414  
EMail: [murphyte@us.ibm.com](mailto:murphyte@us.ibm.com)

Paul F. Rieth  
IBM Corporation  
3605 Highway 52 North  
Rochester, MN 55901

Phone: (507) 253-5218  
Fax: (507) 253-5156  
EMail: [rieth@us.ibm.com](mailto:rieth@us.ibm.com)

Jeffrey S. Stevens  
IBM Corporation  
3605 Highway 52 North  
Rochester, MN 55901

Phone: (507) 253-5337  
Fax: (507) 253-5156  
EMail: [jssteven@us.ibm.com](mailto:jssteven@us.ibm.com)

## Full Copyright Statement

Copyright (C) The IETF Trust (2006).

This document is subject to the rights, licenses and restrictions contained in BCP 78 and at [www.rfc-editor.org/copyright.html](http://www.rfc-editor.org/copyright.html), and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST, AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at [ietf-ipr@ietf.org](mailto:ietf-ipr@ietf.org).

## Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

